INITIATION AU BASIC 128 TO7-70

François-Marie Blondel



Dans la même collection

Initiation au BASIC TO 7 — *Christine et François-Marie Blondel.*
Un ordinateur en fête — *Serge Pouts-Lajus.*
Un ordinateur et des jeux — *Jean-Pascal Duclos.*
Un ordinateur à la maison — *Jean Delcourt*
Manuel Technique du TO 7 et TO 7-70 — *Michel Oury*
Guide du MO 5 — *André Deledicq*
Initiation à LOGO — *Doris Avram - Michèle Weidenfeld*
LOGO, Manuel de référence — *Doris Avram - Tristan Savatier - Michèle Weidenfeld*
Manuel de l'Assembleur 6809 sur TO7/TO7-70 — *Michel Weissgerber*
Initiation au FORTH sur TO7/TO7-70 et MO5 — *S.E.F.I.*

Ce volume porte la référence 2-7124-1714-3

CEDIC, 6-10, boulevard Jourdan. 75014 - PARIS

Première partie

Initiation

Chapitre 1
Qu'est ce qu'une instruction ?
Ecrire à l'écran

Lorsque vous allumez le micro-ordinateur après le téléviseur, le menu vous propose le choix entre:
— le Basic 128
— le réglage du crayon optique

Le Basic 128 est entièrement en mémoire morte. Choisissez donc l'option Basic; l'interpréteur prend le contrôle de l'ordinateur et vous disposez immédiatement de toutes les possibilités offertes par le langage Basic.

La touche ENTREE

Sur le clavier, une touche saute aux yeux, la touche ENTREE.
C'est l'instrument du dialogue avec l'ordinateur et comme dans tout dialogue, vous serez tantôt émetteur, tantôt récepteur.
Quel que soit le message que vous affichiez à l'écran, c'est-à-dire quoi que vous tapiez sur le clavier, ce message restera lettre morte pour l'ordinateur tant que vous ne lui avez pas dit "à vous". Et lui dire

“à vous”, c’est appuyer sur la touche ENTREE .
Seules quelques touches peuvent être comprises directement par l’ordinateur, comme la touche RAZ qui permet d’effacer l’écran.

Lorsque vous appuyez sur la touche ENTREE, votre message est transmis à l’“interpréteur Basic” qui, comme son nom l’indique, est sommé d’essayer de comprendre ce que vous avez écrit. Il ne le pourra que si votre message est composé d’“instructions” qu’il connaît.
Ces instructions ont une syntaxe très stricte (la grammaire constitue la partie Référence de cet ouvrage): elles comprennent un ou plusieurs mots clés et des informations variables.
Voyons ceci sur l’instruction “SCREEN”.

Changer les couleurs de l’écran: SCREEN

Comme vous le savez, votre micro-ordinateur possède seize couleurs différentes. Au départ, l’affichage se fait en bleu foncé sur cyan (bleu pâle). Pour changer les couleurs de l’écran, l’ordre doit en être donné à l’ordinateur sous la forme suivante:

```
SCREEN 7,0,4
```

Tapez cette instruction puis demandez à l’ordinateur de l’“exécuter” avec la touche ENTREE .
Si votre message est bien transmis, et bien reçu..., l’instruction s’exécute et les caractères deviennent blancs sur fond noir, avec un pourtour bleu.
Ensuite l’ordinateur répond OK et le curseur revient au début de la ligne suivante, en attente d’une nouvelle instruction.

L’instruction SCREEN (=écran) est précisée par trois nombres:
— le premier pour la couleur des caractères
— le deuxième pour la couleur du fond
— le troisième pour la couleur du cadre qui entoure le rectangle utile.
Les couleurs initiales sont désignées par des nombres de 0 à 15:

0 noir	6 cyan (bleu pâle)	12 bleu clair
1 rouge	7 blanc	13 magenta clair
2 vert	8 gris	14 cyan clair
3 jaune	9 rouge clair	15 orange
4 bleu	10 vert clair	
5 violet (magenta)	11 jaune clair	

La syntaxe d'une instruction

Une instruction peut souvent s'écrire sous des formes plus ou moins simplifiées.
Ainsi dans l'instruction SCREEN, vous n'êtes pas obligé de faire suivre le mot clé par trois nombres:
— pour changer seulement la couleur des caractères en rouge, tapez

```
SCREEN 1
```

— pour changer la couleur des caractères et du fond, tapez

```
SCREEN 1,5
```

— pour changer seulement la couleur du fond

```
SCREEN ,7
```

La virgule indique que 7 est le deuxième nombre dans l'instruction et représente donc la couleur du fond.
— il existe même un quatrième paramètre (ou "argument") utilisable:

```
SCREEN ,,,1
```

inverse les couleurs des caractères et du fond ("vidéo inverse"). Si vous exécutez une deuxième fois:

```
SCREEN ,,,1
```

il se produit une nouvelle inversion qui ramène l'affichage aux couleurs initiales.

Ces règles constituent la syntaxe de l'instruction SCREEN. Par la suite, nous ne donnerons que la forme la plus courante des instructions, ou celle dont nous aurons besoin, et c'est dans la partie Référence que vous trouverez toutes les possibilités de chaque instruction.

Les messages d'erreur

Au cours de vos premiers essais de dialogue ou en essayant les diverses couleurs dans l'instructions SCREEN, vous avez pu recevoir une réponse comme:

```
Syntax Error
```

C'est que l'interpréteur a reconnu une erreur de syntaxe dans votre instruction, par exemple une faute de frappe dans le mot SCREEN. Si vous donnez un nombre supérieur à 15 pour une couleur, par exemple en oubliant une virgule entre la couleur des caractères et celle du fond, vous obtenez un autre message d'erreur:
Illegal Function Call
ce qui signifie que l'instruction ne peut pas être exécutée avec la valeur donnée.
Vous trouverez la liste de tous les messages d'erreur avec leur signification en Annexe 7.

L'ordinateur calculatrice et machine à écrire: PRINT

L'instruction PRINT permet d'utiliser l'ordinateur à la fois comme calculatrice et comme machine à écrire.

Par exemple:

```
PRINT 60* 24    (toujours suivi par ENTREE)
1440
```

donne le résultat de la multiplication 60x24.
Les quatre opérations sont utilisables comme sur une calculatrice:
+,–,*,/

```
PRINT 72.5/2
36.25
```

N'oubliez pas que le point remplace la virgule dans un nombre décimal.
Il y a d'autres ressources en matière de calculs que nous verrons plus loin.

Pour imprimer du texte à l'écran, il suffit de placer ce texte entre guillemets dans l'instruction PRINT:
PRINT "nom, prénom"
nom, prénom

```
PRINT "999-99-99"
999-99-99
```

Du texte et des calculs peuvent se suivre dans une même instruction PRINT, séparés par des points-virgules. Dans ce cas les affichages se font les uns derrière les autres . Cela permet de préciser le sens des calculs réalisés:

PRINT “nombre de minutes dans un jour:” ; 60*24
nombre de minutes dans un jour: 1 440

```
PRINT 60*24; 60*60*24
1 440  86 400
```

(Bien sûr, pour que l’ordinateur obéisse à vos instructions, il ne faut pas oublier d’appuyer sur ENTREE après chaque instruction...)

Et enfin pour simplifier l’écriture, le mot PRINT peut être remplacé par ? :

```
?12*52
624
? "bonjour"
bonjour
```

Corriger et modifier: l’éditeur de texte

Les touches sur la partie droite du clavier servent à “l’éditeur” de l’ordinateur, c’est-à-dire aux outils permettant de modifier un texte à l’écran.
Que ce soit pour remplacer un caractère par un autre, en effacer ou en ajouter, il faut d’abord placer le curseur à l’endroit à modifier.
Les quatre flèches ←→ ↑ ↓ permettent de placer le curseur en n’importe quel point de l’écran, quant à la touche ⮌ elle remonte le curseur en haut à gauche de l’écran.
Une fois le curseur placé sous le caractère voulu:
— pour remplacer ce caractère par un autre, tapez le nouveau caractère à la place de l’ancien
— pour effacer le caractère , appuyez sur la touche EFF,
— pour effacer le caractère juste à gauche, appuyez sur la touche DEL,
— pour ajouter un ou plusieurs caractères, appuyez sur la touche INS . Le caractère devant lequel va se faire l’insertion s’affiche en vidéo inverse. Tapez les caractères à ajouter, puis appuyez à nouveau sur la touche INS pour sortir du “mode insertion” : le caractère devant lequel s’est faite l’insertion revient aux couleurs standard.

Changer la couleur de l'affichage: COLOR

Pour mettre un ou plusieurs mots en relief sur une page à l'écran, il est possible de changer la couleur des caractères de ces mots.
Tapez:

```
COLOR 1,4
```

Si vous appuyez sur une touche quelconque, le caractère s'affiche en rouge (1) sur fond bleu (4).
L'instruction COLOR a une syntaxe proche de celle de SCREEN puisque le premier chiffre désigne la couleur des caractères et le deuxième celle du fond. Mais elle n'agit que sur les affichages ultérieurs alors que SCREEN agit sur tout l'écran.

Changer la taille des caractères: ATTRB

Si la presbytie vous menace, ou si vous voulez lire votre ordinateur à distance, l'instruction ATTRB est pour vous. Tapez:

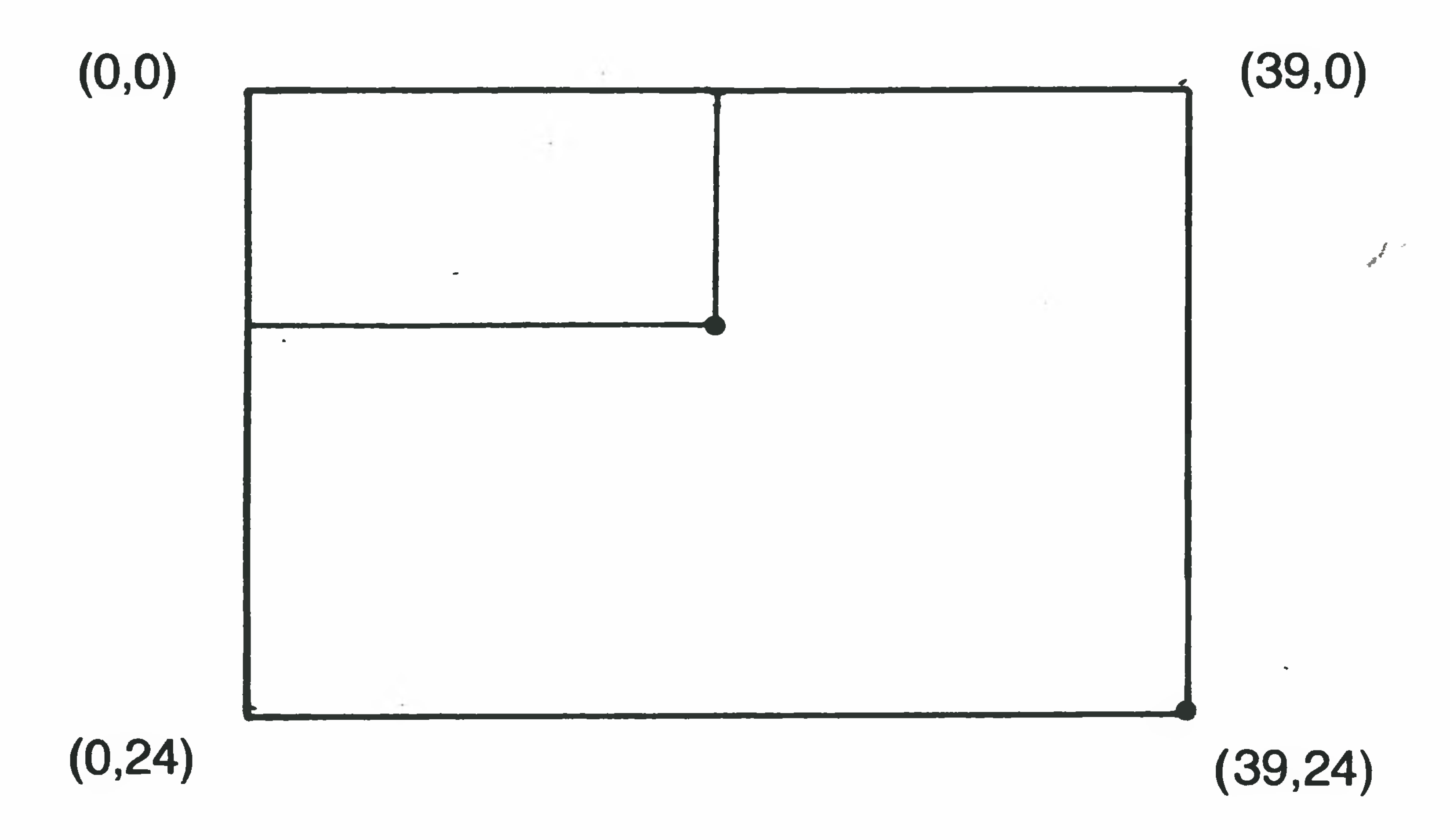
```
ATTRB 1,1: PRINT "ME VOYEZ-VOUS?"
```

Les deux-points entre les deux instructions ATTRB et PRINT servent à séparer deux instructions sur la même ligne: la première instruction précise que les caractères seront deux fois plus grands, la deuxième demande l'affichage.

Changez ATTRB 1,1 en ATTRB 0,1 en remontant le curseur sur la ligne de l'instruction et en remplaçant le premier 1 par un 0, puis appuyez sur ENTREE pour valider la nouvelle instruction. Il est inutile que le curseur soit en fin de ligne pour la valider: chaque appui sur la touche ENTREE demande systématiquement l'exécution de la ligne où se trouve le curseur.
Essayez maintenant ATTRB 1,0 ...

Imprimer où l'on veut à l'écran: LOCATE

L'écran, ou plutôt la "fenêtre" (le rectangle utile), contient 40 caractères par ligne et 25 lignes. Les colonnes sont numérotées de 0 à 39 et les lignes de 0 à 24.



Si vous voulez imprimer BONJOUR vers le milieu de l'écran, nettoyez l'écran avec RAZ, puis tapez:

```
LOCATE 17,12: PRINT "BONJOUR"
```

L'affichage commence sur le caractère situé à l'intersection de la colonne numéro 17 (donc la 18e) et de la ligne numéro 12 (donc la 13e).
Ainsi dans l'instruction LOCATE le premier paramètre précise la colonne et le second la ligne de l'endroit où doit se placer le curseur. Le premier affichage qui suit se fait à cet endroit.

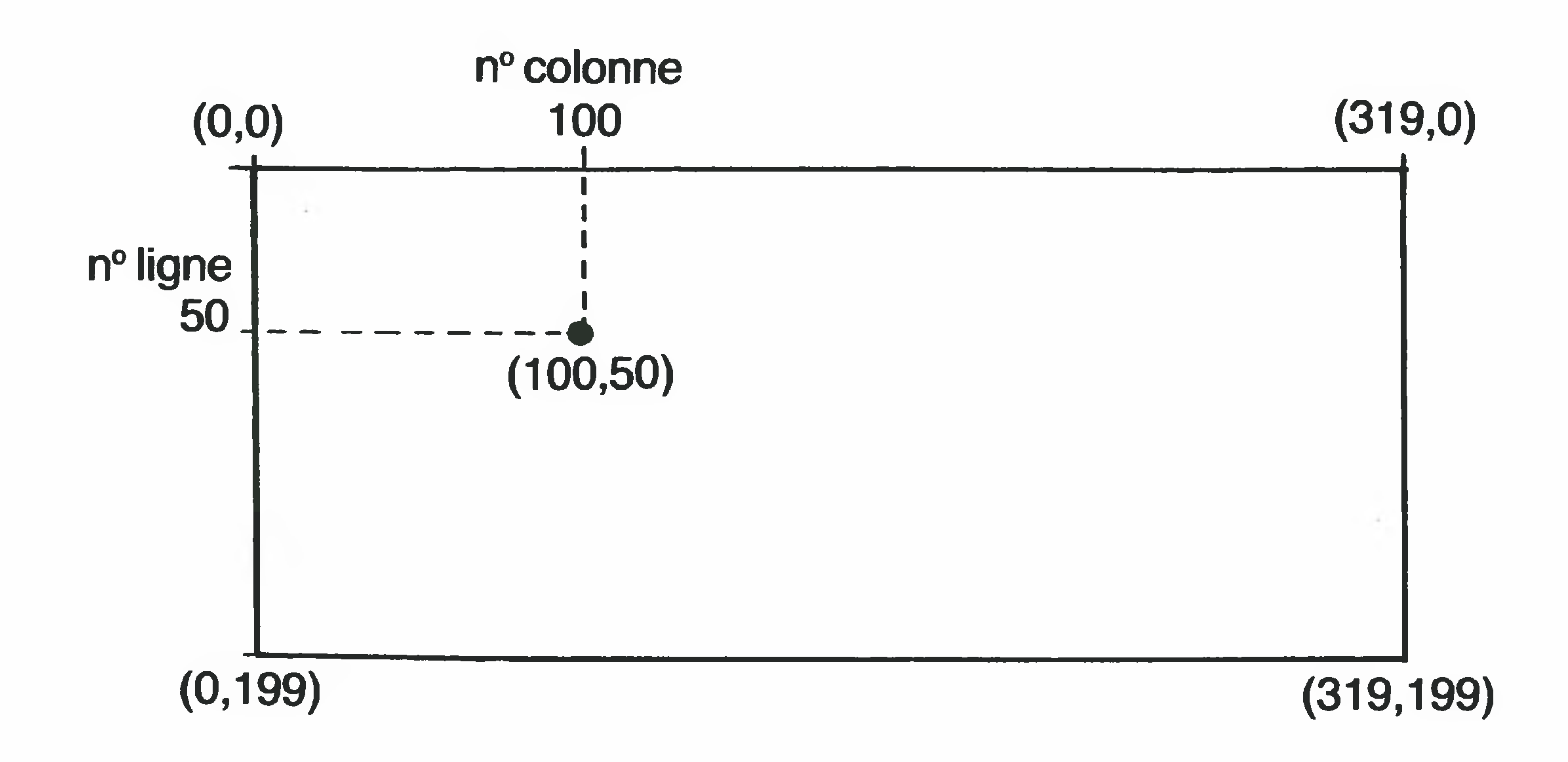
Instructions vues:

SCREEN	COLOR
PRINT	LOCATE
ATTRB	

Chapitre 2
Le graphique
Qu'est-ce qu'un programme ?

Le dessin à l'écran : mode graphique

Nous avons vu que l'écran est divisé en 40 colonnes et 25 lignes lorsqu'il sert à écrire du texte. C'est le "mode caractères"
Pour dessiner, ce mode n'est pas assez précis. Le mode graphique, lui, divise l'écran en 320 colonnes et 200 lignes, ce qui définit 64000 points (ou "pixels") différents et accessibles individuellement.
Le mode graphique est directement utilisable avec les instructions graphiques.
Comme pour le mode caractères, les lignes et colonnes sont numérotées à partir de 0: de 0 à 199 pour les lignes, de 0 à 319 pour les colonnes.



Dans toutes les instructions relatives au graphique, un point est désigné par son numéro de colonne suivi de son numéro de ligne, les deux séparés par une virgule et le tout entre parenthèses: (colonne,ligne).
Ainsi (100,50) désigne le point à l'intersection de la colonne 100 et de la ligne 50.

Tracer un trait: LINE

Pour joindre deux points, par exemple (40,60) et (300,180), on écrit:

```
LINE (40,60)-(300,180)
```

ou de manière équivalente:

```
LINE (300,180)-(40,60)
```

Ainsi pour tracer la ligne horizontale numéro 80:

```
LINE(0,80)-(319,80)
```

Et pour que sa couleur soit différente de celle des caractères (rouge par exemple), on la précise à la fin, séparée par une virgule:

```
LINE(0,80)-(319,80),1
```

Un programme de dessin

Jusqu'ici nous avons utilisé l'ordinateur en mode "direct", c'est-à-dire qu'en tapant une ou plusieurs instructions sur une ligne on en obtient l'exécution immédiate avec ENTREE.
Ceci est assez limité...
Pour faire mémoriser puis exécuter une série d'instructions les unes à la suite des autres, on écrit un programme.
Dans un programme, les instructions doivent être précédées d'un numéro. Leur exécution est différée: lorsque vous appuierez sur la touche ENTREE, l'ordre ne sera pas exécuté.
Si par exemple nous voulons un programme qui, après avoir effacé l'écran, trace une ligne verticale et une ligne horizontale se coupant au milieu de l'écran :

```
10 CLS
20 LINE(0,100)-(319,100)
30 LINE(160,0)-(160,199)
```

Notre programme graphique commence par une instruction équivalente à la touche RAZ: CLS (CLear Screen =nettoyer l'écran).
— Les lignes du programme sont numérotées dans l'ordre où elles doivent s'exécuter, en général de 10 en 10 pour pouvoir intercaler de nouvelles lignes.
— A la fin de chaque ligne, l'appui sur la touche ENTREE ne demande pas l'exécution immédiate mais seulement la "mise en mémoire" dans l'ordinateur de la ligne.
— L'exécution du programme est indépendante de son écriture (c'est-à-dire de sa mise en mémoire) et se fait d'un bloc avec une commande spéciale: RUN (vas-y).

Tapez chaque ligne en la validant par ENTREE, puis demandez l'exécution du programme avec:

```
RUN
```

La colonne numéro 160 et la ligne numéro 100 se tracent à l'écran. Le programme lui-même n'est plus visible à l'écran, mais il est conservé en mémoire, contrairement à ce qui se passe en mode direct.

Afficher le programme (LIST) et le modifier

Pour retrouver cette liste des instructions qui est toujours en mémoire, tapez:

```
LIST
```

Le texte du programme s'affiche à l'écran.

Le programme n'est pas clos: vous pouvez ajouter d'autres lignes à volonté.
Pour tracer les deux diagonales de l'écran, ajoutons les deux lignes suivantes:

```
40 LINE(0,0)-(319,199)
50 LINE(319,0)-(0,199)
RUN
```

Le programme étant en mémoire, vous pouvez par ailleurs demander l'exécution d'autres instructions (sans numéro de ligne) indépendantes du programme, c'est-à-dire travailler en mode direct:

```
PRINT 5.85   2
```

En outre, le programme n'a pas besoin d'être affiché à l'écran pour être exécuté:

```
RAZ
RUN
```

Le programme est bien en mémoire!

La construction d'un programme implique la correction d'erreurs... L'instruction LIST est alors indispensable .
Si par exemple vous avez tapé en ligne 50:

```
50 LINE (319,0)- 0,199)
```

Lorsque vous demandez l'exécution avec RUN , vous obtenez:
Syntax Error in 50
Tapez alors

```
LIST
```

Sur la ligne 50 le caractère fautif est mis en évidence par un pavé de la couleur des caractères:

```
50 LINE(319,0)- ■ 0,199)
```

Dans un programme plus important, vous obtiendrez directement la ligne où se trouve l'erreur en tapant:

```
LIST .
```

C'est le simple point après LIST qui fait la différence...

Des cases bien étiquetées: les variables

L'ordinateur peut aussi conserver des nombres en mémoire, en sachant à quoi ils correspondent. Pour cela, il faut leur donner un nom, ce sont alors des variables.
Prenons le cas d'un prix:

```
PRIX=1000
```

Cette instruction, appelée instruction d'affectation, réserve une zone en mémoire, lui donne le nom PRIX et y inscrit le nombre 1000. Elle définit la variable PRIX et lui affecte une valeur.
Vous pouvez le vérifier en demandant le contenu de la variable:

```
? PRIX
1000
```

Il vous est possible de garder ainsi en mémoire tous les nombres qui vous intéressent; il suffit de donner un nom différent à chacun d'eux:

```
REDUCTION = 0.20
```

L'instruction d'affectation peut se lire:
"la variable REDUCTION reçoit la valeur 0.20".
Il faut bien comprendre que ce n'est pas une égalité comme en mathématiques. Cette instruction se lit dans un seul sens: le nom de la variable s'écrit toujours à gauche, la valeur qu'elle reçoit toujours à droite du signe =.
Pour changer la valeur d'une variable, il faut lui affecter une nouvelle valeur:

```
REDUCTION = 0.25
?REDUCTION
.25
```

L'ancienne valeur 0.20 est effacée.

La variable une fois définie peut être utilisée dans des calculs ou dans d'autres instructions. On peut par exemple définir le montant de la remise:

```
MONTANT=PRIX*REDUCTION
?MONTANT
250
```

La nouvelle variable MONTANT prend la valeur calculée par le produit de la variable PRIX (1000) avec la variable REDUCTION (0.25).
On peut alors calculer le nouveau prix, c'est-à-dire la nouvelle valeur de la variable PRIX:

```
PRIX=PRIX–MONTANT
?PRIX
750
```

La valeur d'une variable peut donc être utilisée pour calculer la nouvelle valeur à lui affecter. Le Basic commence par calculer cette valeur à partir des variables de la partie droite puis l'affecte à la variable de la partie gauche.

Les variables peuvent aussi être utilisées dans des instructions qui ne font pas de calcul:

```
COLONNE = 100: LIGNE = 100
LINE (COLONNE,LIGNE)–(160,100)
```

Le point de départ du tracé est bien en (100,100).
Essayez maintenant:

```
LINE (X,Y)-(160,100)
```

Cette fois, le tracé part du point (0,0)pour aller en (160,100).
En effet, une variable à laquelle aucune valeur n'a été affectée, prend a priori la valeur 0. Donc X et Y valent 0 jusqu'à la première affectation.

Les noms de variables peuvent être des mots quelconques, contenant des chiffres, mais ils doivent commencer par une lettre:

```
PRIXDUCAFE = 4.50
COULEUR2 = 15
HLAB27 = 0
```

Une petite restriction, ils ne doivent pas commencer par un mot clé du Basic (SCREEN, COLOR, ...). La liste de ces mots est donnée en Annexe 5.
Ainsi,

```
VALEUR=PRIX*REDUCTION
```

vous retourne le message "Syntax Error" car VALEUR commence par VAL qui est un des mots clés du Basic.

Un programme qui calcule

Lorsqu'on dispose d'un moyen de transport, il peut être utile de calculer sa vitesse moyenne pour un trajet donné.
Soit par exemple une durée de 4 h 30 pour un trajet de 294 kilomètres.
Ecrivons le programme qui calcule la vitesse moyenne du parcours:
— Le programme précédent étant toujours en mémoire, il faut d'abord l'effacer. Sinon, les lignes d'instructions vont se mélanger.
Pour cela, tapez la commande NEW (nouveau):

```
NEW
```

Vous pouvez vérifiez avec LIST que la mémoire ne contient plus d'instructions.

— Dans la première ligne, donnons un nom au programme. Il est courant de donner aussi la date d'écriture, le nom de l'auteur... Tous

les commentaires sont possibles dans une ligne de programme à condition de faire précéder ces commentaires par REM (remarque) ou une apostrophe:

```
10 REM CALCUL DE VITESSE
```

ou de manière équivalente:

```
10 ' CALCUL DE VITESSE
```

— Plaçons les deux données, temps et distance, dans des variables:

```
20 T=4.5
30 D=294
```

Si les Babyloniens affectionnaient le système sexagésimal (à base 60), qu'ils nous ont légué pour la mesure du temps, le langage Basic ne connaît que le système décimal. L'ordinateur, lui, utilise le système binaire, mais nous n'en aurons pas besoin.
Ainsi 4 h 30 doit s'écrire 4,50 ou plutôt 4.5 .

— Demandons l'affichage du résultat du calcul .
La vitesse moyenne est D/T, et elle est exprimée en km/h:

```
40 PRINT "VITESSE";D/T;"KM/H"
```

Le programme est donc :

```
10 'CALCUL DE VITESSE
20 T=4.5
30 D=294
40 PRINT "VITESSE";D/T;"KM/H"
```

— Demandons l'exécution :

```
RUN
VITESSE 65.3333 KM/H
```

La division ne tombe pas juste et l'ordinateur a "arrondi" le résultat en ne gardant que six chiffres. Il peut cependant en garder jusqu'à seize si vous le voulez (voir chapitre 5, la notation des nombres et leur précision).

Ajouter des lignes au programme

Si pour un troisième trajet la durée est 4 h 11, on peut envisager de demander à l'ordinateur de faire la conversion dans le système décimal!

Appelons H le nombre d'heures et MN celui des minutes. Le temps en heures sera: T = H + MN/60 .
Les nouvelles variables H et MN sont introduites dans le programme dans deux lignes supplémentaires insérées entre la ligne 10 et la ligne 20:

```
14 H=4
16 MN=11
```

et la ligne 20 est modifiée:

```
20 T=H+MN/60
```

Dans la mémoire, les lignes sont automatiquement remises dans l'ordre de leurs numéros. Ces numéros peuvent aller jusqu'à 64000...
Avant de demander l'exécution, il est toujours prudent de vérifier que les modifications sont bien en place:

```
LIST
10 'CALCUL DE VITESSE
14 H=4
16 MN=11
20 T=H+MN/60
30 D=294
40 PRINT"VITESSE";D/T;"KM/H"

RUN
VITESSE 70.2789 KM/H
```

Interroger l'utilisateur: INPUT

Notre programme de calcul de vitesse doit être modifié pour chaque nouveau calcul, ce qui suppose que l'utilisateur du programme est lui-même programmeur.
Mais l'ordinateur ne sait pas seulement répondre, il peut aussi interroger l'utilisateur et travailler avec ses réponses. Ainsi nous allons lui faire demander combien d'heures et de minutes ont été nécessaires pour effectuer le trajet.
L'instruction qui permet de poser une question et d'enregistrer la réponse est INPUT (=entrée).

Modifions les lignes 14 et 16 et ajoutons les lignes 15 et 17:

```
14 PRINT "COMBIEN D'HEURES"
15 INPUT H
16 PRINT "COMBIEN DE MINUTES"
17 INPUT MN
```

le reste du programme restant identique.

A l'exécution, l'ordinateur affiche:

```
COMBIEN D'HEURES
?
```

A la ligne 15 l'instruction INPUT affiche un point d'interrogation et attend une réponse. L'exécution du programme est interrompue et vous devez taper un nombre, par exemple 4.
Ensuite l'exécution du programme reprend et s'affiche:

```
COMBIEN DE MINUTES
?
```

Vous devez encore donner un nombre, par exemple 11.
Alors l'exécution reprend à la ligne 20 et le résultat du calcul de votre vitesse s'affiche à l'écran.C'est le même que dans le programme précédent.

Attention, vos réponses doivent impérativement être des nombres. Si vous tapez une ou plusieurs lettres, vous obtenez comme réponse:

```
?Redo from start
```

(Recommencez...)

INPUT permet aussi d'imprimer

Pour que le point d'interrogation s'affiche à la fin de la question, ajoutons un point-virgule à la fin des lignes 14 et 16:

```
14 PRINT "COMBIEN D'HEURES";
16 PRINT "COMBIEN DE MINUTES";
LIST
RUN
```

Ce résultat peut être obtenu encore plus simplement avec la seule instruction INPUT:

```
14 INPUT "COMBIEN D'HEURES";H
16 INPUT "COMBIEN DE MINUTES";MN
```

L'instruction INPUT permet donc à la fois d'afficher le texte de la question et d'enregistrer la réponse.
Les lignes 15 et 17 sont alors inutiles. Pour les supprimer il suffit d'entrer deux lignes vides à la place:

```
15
17
LIST
```

Vérifiez que le programme donne le même résultat:

```
RUN
```

On peut encore demander les valeurs de H et MN en une seule question:

```
14 INPUT "HEURES,MINUTES";H,MN
```

et supprimer la ligne 16:

```
16
LIST
RUN
```

Lors de l'exécution, la question est:

```
HEURES,MINUTES?
```

Il faut alors répondre dans l'ordre, en séparant les deux nombres par une virgule:

```
4,11
```

Instructions vues:

LINE
CLS
RUN
LIST
NEW
REM

Chapitre 3
Répétition et graphiques

Modifier l'ordre de l'exécution: GOTO

Souvent dans un programme, une même action est répétée un grand nombre de fois. Voyons d'abord le cas le plus simple, celui de la répétition à l'infini.
Pour tracer des droites verticales à l'écran, par exemple de 10 en 10, il faut répéter la même instruction "tracer une colonne", le numéro de la nouvelle colonne étant égal à celui de la précédente augmenté de 10 . Plaçons le numéro de la colonne à tracer dans une variable: COL.
Le cœur du programme sera:
— tracer la colonne numéro COL
— augmenter COL de 10 unités
— recommencer
Ceci est une boucle sans fin.
L'instruction qui permet de faire remonter l'exécution du programme à une ligne précédente, ou de lui faire sauter plusieurs lignes en avant, est GOTO (va) suivie du numéro de la ligne où aller.
D'où la boucle en Basic:

```
40 LINE (COL,0)-(COL,199)
50 COL=COL+10
60 GOTO 40
```

Il faut encore donner une valeur initiale à COL (valeur de la première colonne tracée) par exemple 10:

```
10 'VERTICALES
20 CLS
30 COL=10
40   LINE (COL,0)-(COL,199)
50   COL=COL+10
60   GOTO 40
RUN
```

L'indentation au début des lignes 40, 50 et 60 facilite la lecture en détachant un bloc d'instructions.

Lorsque la dernière droite verticale a été tracée, le programme semble s'arrêter. Il n'en est rien. Si vous appuyez sur une touche, rien ne s'affiche à l'écran: ceci est caractéristique de l'ordinateur en train de tourner (ici en rond...).
Pour arrêter l'exécution, appuyez sur la touche CNT (Control) et en gardant la pression sur cette touche, appuyez sur C.
Vous récupérez le contrôle et pouvez demander quelle est la valeur de la variable COL, donc celle atteinte lorsque le programme s'est arrêté:

```
?COL
```

Le nombre obtenu est très supérieur à 319. Si vous recommencez l'exécution et attendez quelques dizaines de secondes, le programme finit par s'arrêter de lui-même et fournit le message d'erreur:

```
Overflow in 40
```

Vous avez dépassé une valeur limite dans une instruction de la ligne 40. En effet l'instruction LINE accepte des numéros de colonnes allant jusqu'à 32767 vers le bas et remontant jusqu'à −32768 vers le haut. Et de même pour les lignes.
Demandez la valeur de la variable COL atteinte lorsque le programme s'est arrêté:

```
?COL
32770
```

Au tour suivant de la boucle, l'instruction LINE n'a pu accepter cette valeur et le programme s'est arrêté après avoir effectué plus de 3200 boucles en moins d'une minute.
Ceci donne au passage une idée du rythme de travail de l'ordinateur.

Interrompre la boucle lorsqu'une condition n'est plus remplie: IF...THEN

Il serait souhaitable que le programme précédent s'arrête dès que la variable COL dépasse 319 et donc effectue la boucle seulement si COL reste inférieur à 319, soit en français:

40 tracer la colonne numéro COL
50 augmenter COL de 10 unités
60 si COL est encore inférieur à 319 alors recommencer

L'instruction qui permet de traduire la ligne 60 est IF(si)...THEN (alors)...:

```
60 IF COL<319 THEN GOTO 40
```

Tant que COL est inférieur à 319, la ligne 60 renvoie en ligne 40. Mais dès que la condition COL<319 devient fausse, l'exécution se poursuit à la ligne suivante du programme au lieu de remonter en ligne 40. Comme il n'y a pas de ligne après la ligne 60, le programme s'arrête...

L'écriture peut se simplifier car GOTO est facultatif après THEN:

```
60 IF COL<319 THEN 40
```

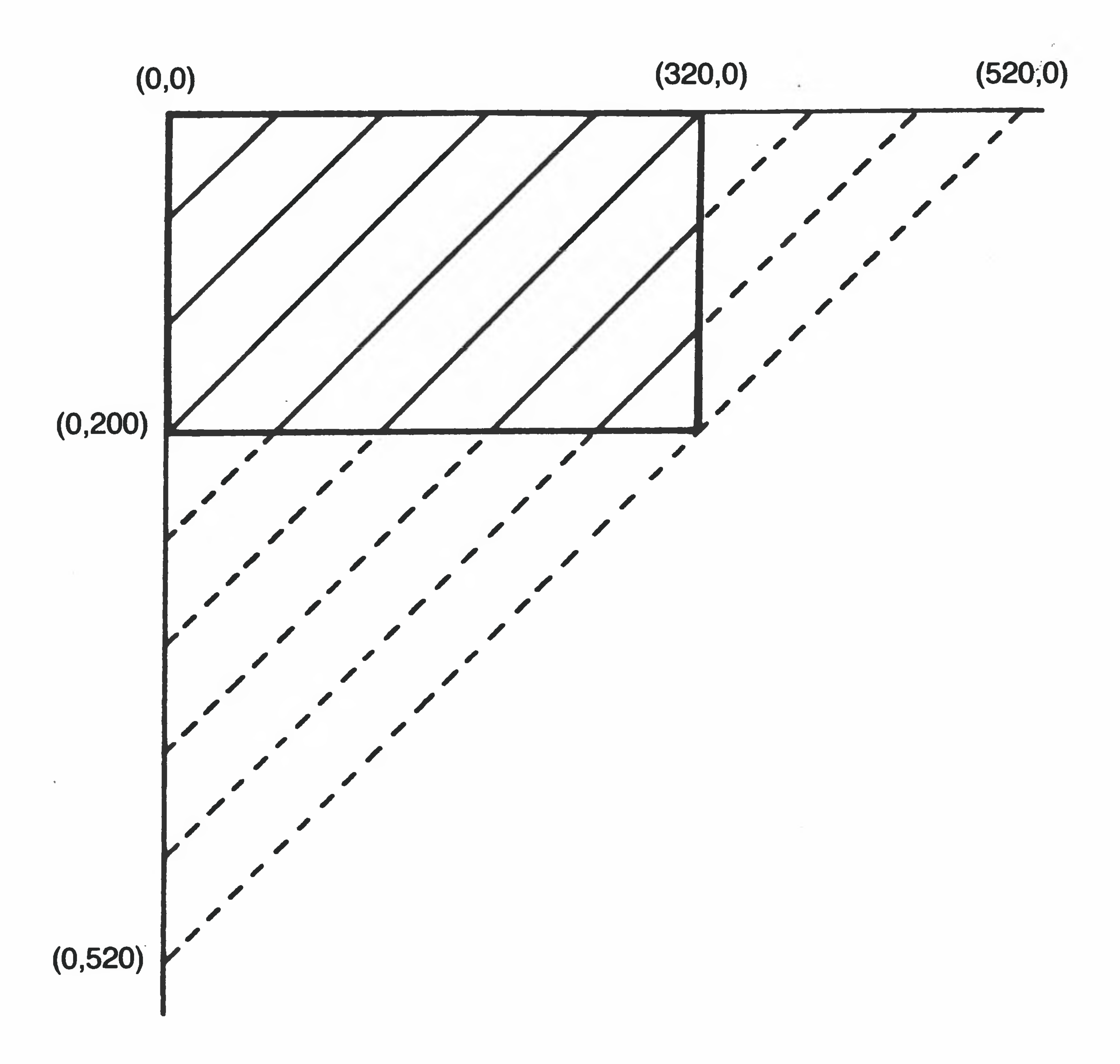
L'instruction IF...THEN permet donc de tester si une condition est vraie ou non et d'effectuer une action différente dans chaque cas:
IF condition THEN instruction
La condition s'exprime par une relation faisant intervenir une ou plusieurs variables (ici COL<319), l'instruction qui suit THEN peut être quelconque (ici GOTO 40).
Quand la condition est vraie, l'instruction est exécutée; quand elle est fausse, l'instruction n'est pas exécutée et on passe à la ligne suivante (quand elle existe).

Utiliser les points en dehors de l'écran...

Si l'instruction LINE (comme toutes les instructions graphiques) accepte des numéros de colonnes ou de lignes allant jusqu'à 32767 cela signifie que vous pouvez utiliser tous les points d'un très grand écran dont seule une petite partie serait visible.

Essayons de modifier le programme précédent pour qu'il trace des barres obliques à 45 degrés.

Si on se limite à l'écran,toutes les barres ne peuvent être définies de la même manière. Ce n'est plus le cas si elles sont prolongées afin de couper les deux axes:



Toutes les barres (ou leurs prolongements) coupent alors les axes à égale distance de l'origine et peuvent donc être tracées avec la même instruction:

```
40 LINE (COL,0)-(0,COL)
```

Le test doit être modifié en conséquence: le programme doit s'arrêter dès qu'une barre est entièrement en dehors de l'écran, c'est-à-dire dès que COL dépasse 200+320=520 .

```
60 IF COL<520 THEN 40
```

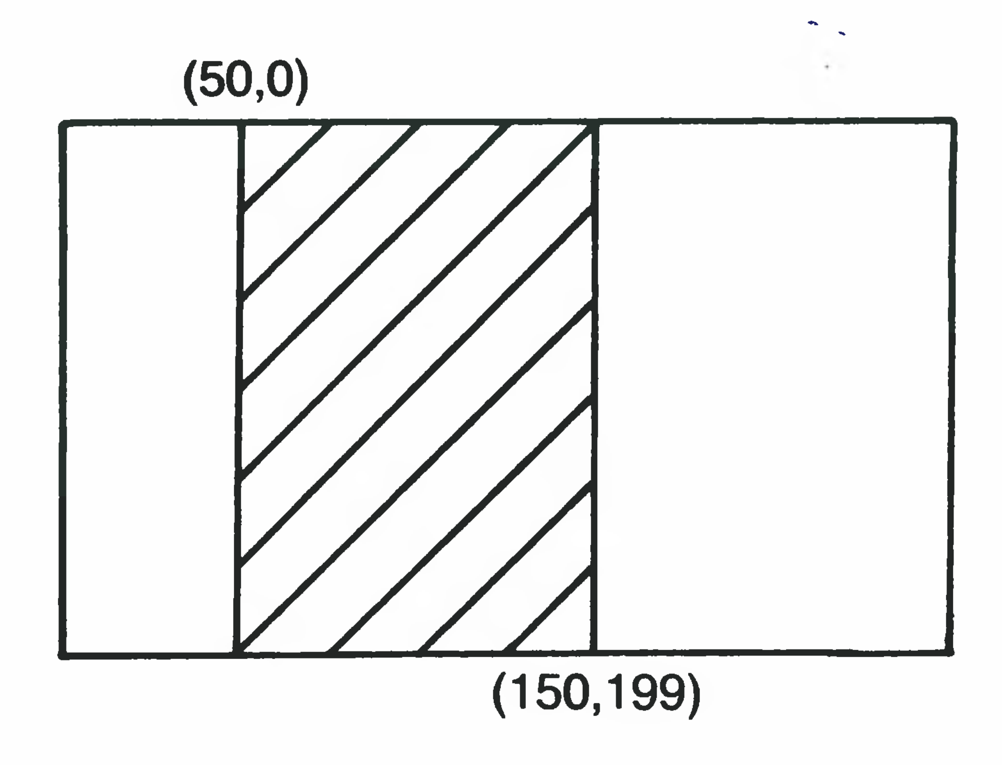
Si vous voulez des barres plus inclinées, modifiez l'intersection avec un des deux axes:

```
40 LINE (COL,0)-(0,2*COL)
```

Visualiser seulement une fenêtre de l'écran: WINDOW

Puisqu'il est possible de dessiner en dehors de l'écran visible, c'est un peu comme si on voyait une partie d'un grand écran (de −32768 à +32767 de chaque côté) à travers une petite fenêtre de la taille de l'écran (de 0 à 319 en colonnes et de 0 à 199 en lignes).
L'instruction WINDOW (fenêtre) permet de réduire encore la taille de la fenêtre et de la localiser où l'on veut dans la partie visible.

Ainsi, si les barres inclinées ne vous intéressent que sur le rectangle suivant:



ajoutez l'instruction suivante au début du programme:

```
25 WINDOW (50,0)-(150,199)
```

Le rectangle-fenêtre est défini par les deux sommets opposés (50,0) et (150,199); le premier sommet est toujours celui situé en haut à gauche et le second celui situé en bas à droite.
Quand vous exécutez le programme, les lignes n'apparaissent plus que dans le rectangle défini par WINDOW. On ne voit plus que par la fenêtre.
Pour revenir à l'état initial, il suffit de faire:

```
WINDOW (0,0)-(319,199).
```

Marquer un point: PSET

Trois instructions LINE permettent de tracer un triangle:

```
10 'TRIANGLE
20 LINE(60,150)-(140,80)
30 LINE-(220,150)
40 LINE-(60,150)
```

Lorsque les segments se suivent "sans lever le crayon", il n'est pas nécessaire de répéter les coordonnées du premier point dans l'instruction LINE, sauf pour la première.
Puisque l'instruction LINE se contente de la 2e extrémité, il faudrait une instruction pour marquer la 1re. Il en existe une qui permet de marquer un seul point: PSET (Point SET: marquer un point).
Le tracé d'un triangle peut alors se réécrire de la façon suivante:

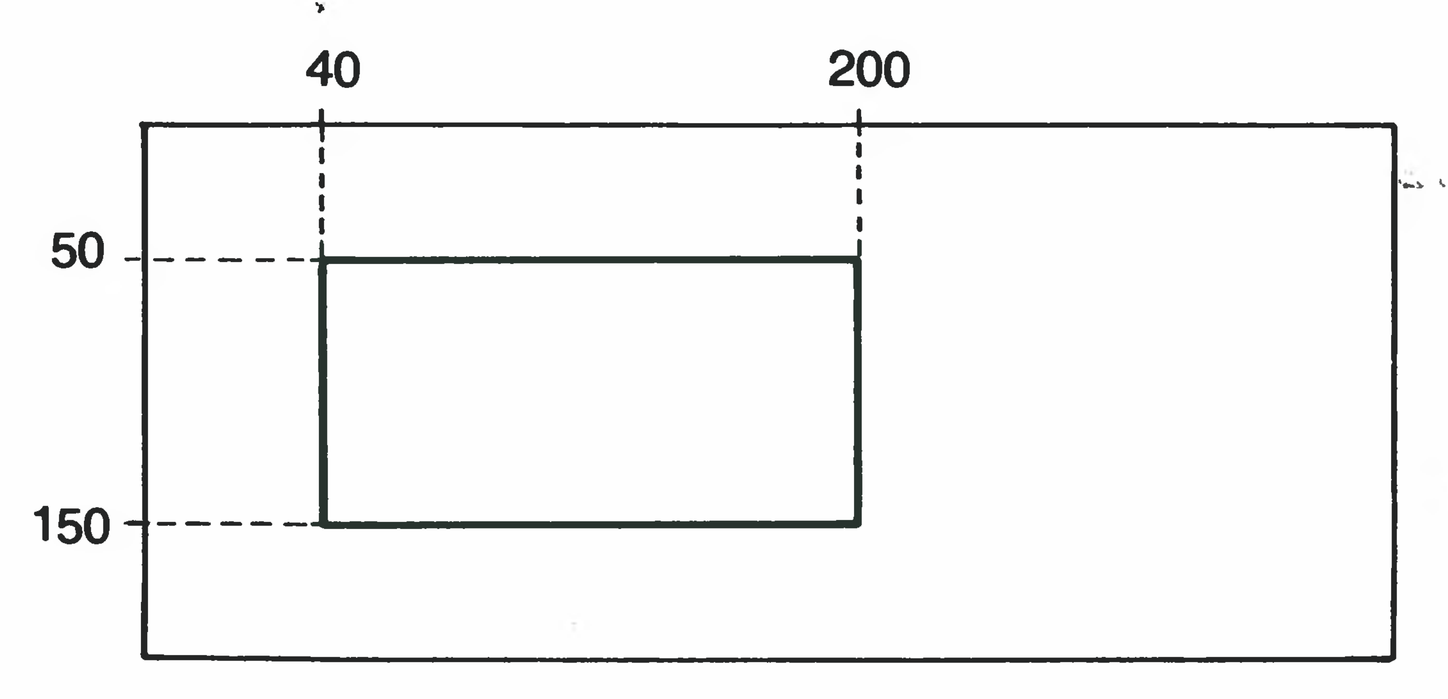
```
10 'TRIANGLE
20 PSET (60,150)
30 LINE -(140,80)
40 LINE -(220,150)
50 LINE -(60,150)
```

Dans PSET, comme dans toutes les instructions graphiques, on indique les coordonnées du point entre parenthèses, colonne en premier, ligne en second.
Et pour changer la couleur du point, on la précise à la fin:

```
PSET (60,150),9
```

Tracer un rectangle: BOX,BOXF

Pour tracer un rectangle, inutile d'écrire quatre instructions LINE. Une seule instruction dessine un rectangle défini par deux sommets opposés, BOX (boîte):



```
BOX (40,50)-(200,150)
```

Le même résultat sera obtenu avec les deux autres sommets du rectangle:

```
BOX (40,150)-(200,50)
```

La couleur, comme pour LINE et PSET, peut être différente de celle des caractères courants:

```
BOX (40,50)-(200,150),10
```

Mieux, la boîte peut être colorée entièrement grâce à une variante de l'instruction, BOXF (BOX FULL: boîte pleine):

```
BOXF (40,50)-(200,150),10
```

Couleur de fond, couleur de forme

Une boîte colorée, c'est une belle toile de fond. Vous pourrez y écrire et y dessiner à condition de respecter les deux types de peintures de votre micro-ordinateur: couleur de fond et couleur de forme.

Traçons une ligne bleue en travers de la boîte pleine précédente:

```
10 CLS
20 BOXF(40,50)-(200,150),10
30 LINE(0,0)-(150,200),4
```

A l'intérieur de la boîte, la ligne se transforme en une succession de petits rectangles...

En effet chaque ligne horizontale à l'écran est divisée en une suite de quarante segments de huit points:

```
0      78      16
........................................
```

A l'intérieur de chaque segment de huit points, il n'y a que deux couleurs possibles: une pour le fond (c'est la couleur des points quand rien n'a été tracé), l'autre pour la forme (c'est la couleur du tracé et des caractères). Si l'on trace deux fois de suite un ou plusieurs points d'un segment, c'est la dernière couleur qui l'emporte.

Couleur de fond et couleur de forme sont déterminées au départ par l'instruction SCREEN (ou COLOR pour les caractères).

Pour que les points de la boîte restent vert clair, il faut qu'ils soient des points "fond". Colorier en couleur de "fond" se fait en donnant un code négatif aux couleurs: −(C+1) pour la couleur de numéro C. Soit −(10+1)=−11 pour le vert clair:

```
20 BOXF(40,50)-(200,150),-11
```

Cette fois plus de problème: les points à l'intérieur de la boîte sont des points "fond" et ne sont plus atteints par le virement de couleur des points "forme" lors du tracé de la droite.

Effacer un dessin

Pour effacer un trait, il faut repasser ses points en couleur de fond, en utilisant les couleurs négatives, afin de pouvoir redessiner ultérieurement sans rencontrer le problème précédent.
Traçons un segment jaune (3) sur fond noir (0):

```
RAZ
SCREEN 7,0,0
LINE (200,150)-(300,150),3
```

puis effaçons-le en rendant ses points au fond (−1 pour le noir):

```
LINE (200,150)-(300,150),-1
```

Si vous faites un programme de cette suite d'instructions en insérant des numéros de lignes:

```
10 CLS
20 SCREEN 7,0,0
30 LINE (200,150)-(300,150),3
40 LINE (200,150)-(300,150),-1
```

l'effaçage est très rapide.

Choisir son motif: PATTERN

Une boîte peut se remplir aussi avec un caractère:

```
10 PATTERN "/"
20 BOXF (40,50)-(200,150)
```

Tous les caractères peuvent servir de motif de remplissage, ce qui laisse nombre de possibilités de décoration. Si vous choisissez comme PATTERN (motif) la barre oblique:

```
10 PATTERN "/"
```

vous retrouvez un rectangle rempli de barres obliques par une tout autre méthode ...

Instructions vues

GOTO	PSET
IF THEN	BOX, BOXF
WINDOW	PATTERN

Pour arrêter le déroulement d'un programme, appuyez simultanément sur les touches CNT et C.

Chapitre 4
Les boucles : forme générale

Répéter un nombre de fois connu à l'avance: FOR...NEXT...

Essayons d'afficher des bandes horizontales de couleur. Une bande sera obtenue très simplement en imprimant une série d' "espaces" dont la couleur de fond est modifiée par une instruction COLOR.
Si nous voulons voir toutes les couleurs à l'écran, nous savons d'avance que cette même action sera répétée exactement seize fois.

Par exemple pour une bande rouge:

```
COLOR,1:PRINT"            "
```

L'instruction COLOR ne modifie ici que le fond et laisse inchangée la couleur des caractères, l'instruction PRINT affiche ensuite douze "blancs" de couleur rouge...
Et pour revenir à un fond noir:

```
COLOR,0
```

Pour écrire la boucle qui va tracer une bande à l'écran, nous pourrions utiliser l'instruction GOTO et tester avec IF...THEN le numéro de la couleur (placé dans la variable C).
Mais quand le nombre de répétitions est connu à l'avance, il est plus simple (et plus facilement modifiable) d'écrire:
— pour C allant de 0 à 15
— afficher la bande de couleur C
— couleur suivante
Ceci se fait avec l'instruction FOR...NEXT:

```
10 'COULEURS
20 CLS: SCREEN 6,0,0
30 FOR C=0 TO 15
40 COLOR ,C: PRINT"  "
50 NEXT C
60 COLOR ,0
```

La ligne 30 signifie "pour C variant de 0 à 15, faire ce qui suit jusqu'à l'instruction NEXT".
La ligne 50 augmente la variable C de 1 (NEXT: suivant) et renvoie l'exécution en ligne 40 si cette nouvelle valeur est encore inférieure à 15.
Lorsque C atteint la valeur 16, alors l'exécution passe à la ligne qui suit l'instruction NEXT, c'est-à-dire ici la ligne 60 qui remet le fond en noir. Pour le vérifier, ajoutons une ligne qui affiche la valeur atteinte par C:

```
70 PRINT C
```

A l'exécution, la valeur 16 s'affiche dans les couleurs d'origine: cyan sur fond noir.

La boucle formée des lignes 30, 40, 50 est équivalente à celle-ci:

```
30 C=0
40   IF C>15 THEN 60
50   COLOR ,C: PRINT"  "
55   C=C+1: GOTO 40
```

Remarquez que dans cette formulation, le test est situé au début de la boucle, comme c'est le cas pratiquement dans l'instruction FOR...NEXT.

Progresser à grands pas : FOR...NEXT...STEP

Séparons l'écran en deux par une ligne verticale :

```
LINE (160,0)-(160,200)
```

Comment tracer cette ligne en pointillés?
Il faut tracer de petits segments de longueur constante, en les espacant d'une longueur constante. Un petit segment de longueur 5 commencant à la ligne LIG se fait par :

```
LINE(160,LIG)-(160,LIG+5)
```

Pour tracer des segments de 10 en 10, il faut faire progresser la variable LIG de 10 en 10, c'est-à-dire par pas de 10. Traduisons en Basic :

```
10 'POINTILLES
20 CLS
30 FOR LIG=0 TO 200 STEP 10
40 LINE (160,LIG)-(160,LIG+5)
50 NEXT LIG
```

L'effet de "STEP 10" est le suivant : à chaque exécution de NEXT LIG, la variable LIG est augmentée de 10 au lieu de 1.
Vérifions immédiatement après l'exécution du programme :

```
PRINT LIG
210
```

Ainsi une borne supérieure de 190 est suffisante :

```
30 FOR LIG=0 TO 190 STEP 10
```

Le pas de 10 permet de faire de grands pas en avant mais aussi en arrière (STEP −10). Dans ce cas, il ne faut pas oublier d'inverser les valeurs de départ et d'arrêt de LIG :

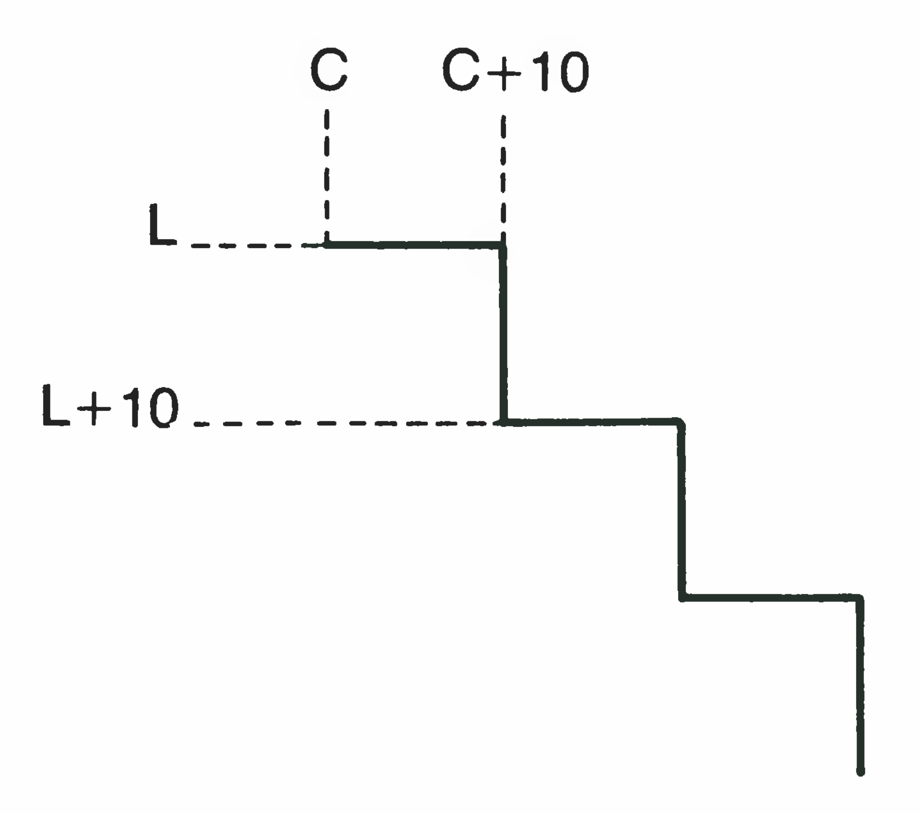
```
30 FOR LIG=190 TO 0 STEP -10
```

La valeur affectée à LIG à la fin du programme est alors −10.

Forme générale d'une itération : DO ... LOOP

La plupart du temps, il n'est pas facile de connaître à l'avance le nombre d'itérations c'est-à-dire le nombre de répétitions. Cepen-

dant, l'écriture d'une boucle est beaucoup plus agréable avec FOR...NEXT qu'avec GOTO. Nous allons voir que l'utilisation du couple d'instructions DO...LOOP apporte la solution.
Prenons un exemple: la construction d'un escalier.
La pièce élémentaire d'un escalier, c'est une marche. Il suffit de la répéter un certain nombre de fois.



En partant du point (colonne C, ligne L) une marche descendante se trace ainsi:

```
C=C+10
LINE-(C,L)
L=L+10
LINE(C,L)
```

Pour répéter le tracé d'une marche, on le place entre les deux instructions:
— début de boucle DO (fait)
— fin de boucle LOOP (boucle)

Voici le programme de l'escalier infini:

```
10 'ESCALIER DESCENDANT
20 CLS
30 C=10: L=10
40 PSET (C,L)
50 DO
60 C=C+10
70 LINE-(C,L)
80 L=L+10
90 LINE-(C,L)
100 LOOP
```

La séquence d'instructions 60 à 90 se répète indéfiniment jusqu'à arrêt par CNT-C ou par débordement de la variable C au-delà de 32768 (cela fait longtemps qu'on ne voit plus rien bouger).
Pour arrêter la boucle (et l'escalier), il faut une instruction de sortie de boucle: EXIT. Le test d'arrêt doit empêcher de descendre trop bas. Ajoutez:

```
95 IF L>200 THEN EXIT
```

et vérifiez le résultat. Après avoir exécuté le programme, C vaut 210 et L vaut 210.
L'instruction EXIT provoque un branchement à l'instruction qui suit LOOP. Ici, comme il n'y a plus rien, le programme s'arrête. Pour changer le niveau inférieur de l'escalier, il suffit de modifier la valeur maximum de L en ligne 95:

```
95 IF L>140 THEN EXIT
```

et pour finir sur une marche horizontale, plaçons la ligne 95 en 75:

```
75 IF L>140 THEN EXIT
```

et supprimons la ligne 95.
Le test de sortie peut donc être placé en n'importe quel endroit de la boucle mais l'effet est tout à fait différent comme vous pouvez le constater.
Pourquoi ne pas faire remonter l'escalier en face? Voici la suite du programme:

```
110 DO
120 C=C+10
130 LINE -(C,L)
140 IF L<10 THEN EXIT
150 L=L-10
160 LINE -(C,L)
170 LOOP
```

Des boucles emboîtées: un chronomètre

Nous allons faire un programme qui transforme l'ordinateur en horloge, c'est-à-dire qui compte régulièrement les secondes, les minutes et les heures.

L'ordinateur, lorsqu'il compte la suite des nombres 1, 2, 3, 4... est parfaitement régulier. Ceci va nous servir de base pour l'horloge: mesurons le temps mis par l'ordinateur pour compter, par exemple de 1 à 10000.
Notre "compteur" aura la forme suivante:
— incrémenter (=augmenter d'une quantité fixée) la variable qui compte: T
— si le compteur T a atteint 10000 sortir de la boucle
— (sinon) recommencer
Ou en Basic:

```
100 T=0
110 DO
120   T=T+1
130   IF T=10000 THEN EXIT
140 LOOP
```

L'exécution de ce programme ne produit rien à l'écran, la variable T a seulement augmenté d'une unité à chaque tour de boucle:

```
?T
10001
```

Recommencez l'exécution du programme en mesurant avec votre montre le temps écoulé entre l'appui sur ENTREE (après RUN) et l'affichage à l'écran de OK. C'est le temps mis par l'ordinateur pour compter de 1 à 10000, soit environ 100 secondes. (Cette valeur peut être légèrement différente sur votre ordinateur.)
En une seconde la variable T passe donc de 0 à 100.

Notre horloge doit compter les secondes et donc pour chaque seconde compter de 1 à 100: une boucle de comptage par seconde.
La partie du programme donnant les secondes aura donc la structure suivante:
— compter pendant une seconde (boucle de comptage de 1 à 100)
— incrémenter le nombre des secondes S
— afficher S
— si S est égal à 60, sortir de la boucle
— (sinon recommencer)

La boucle de comptage du temps se trouve donc à l'intérieur de la boucle de comptage des secondes. Les deux boucles sont "emboîtées" :

```
10 'CHRONO
20 CLS : S=0
80  DO
100  T=0
110    DO
120    T=T+1
130    IF T=100 THEN EXIT
140    LOOP
150  S=S+1
160  LOCATE 10,10: PRINT S
170  IF S=60 THEN EXIT
180  LOOP
```

Nous avons décalé les lignes à l'intérieur de chaque boucle pour mieux les visualiser.
La boucle principale, qui compte les secondes, s'étend du DO de la ligne 80 au LOOP de la ligne 180; on en sort en ligne 170 quand S vaut 60.
La boucle secondaire, ou interne, s'étend de la ligne 110 à la ligne 140.
L'interpréteur Basic fonctionne de la manière suivante:
— quand il rencontre LOOP, il recommence l'exécution à partir du dernier DO rencontré
— quand il rencontre EXIT, il saute les instructions jusqu'à celle qui suit le premier LOOP suivant.
Par la même méthode, on peut ajouter les minutes avec la boucle suivante:

```
40 MN=0
50 DO
190 MN=MN+1
200 LOCATE 5,10: PRINT MN
210 IF MN=60 THEN EXIT
220 LOOP
```

L'exécution fournit une horloge qui compte les minutes et les secondes de 1 à 60 et non pas de 0 à 59, comme il est de mise sur une bonne horloge.

Ceci est dû à l'endroit où sont placées les instructions d'affichage: en fin de boucle. Or l'horloge doit afficher 0 au déclenchement, 1 au début de la première seconde (et non à la fin) ...
Déplaçons la ligne 160 en 90 et la ligne 200 en 60 .
Ainsi l'horloge compte les secondes et minutes en cours et non écoulées.

La place d'une instruction dans une boucle n'est pas indifférente et peut modifier sensiblement l'allure du résultat

Il existe bien d'autres manières de faire un chronomètre. On pourrait par exemple remplacer les boucles DO...LOOP par FOR...NEXT puisqu'on connaît à l'avance la valeur des bornes.
Essayez-le.
Une autre solution sera examinée un peu plus loin avec ON INTERVAL (chapitre 10), elle a le mérite de laisser un autre programme fonctionner en même temps, avec de surcroît une très bonne précision.

Instructions vues

FOR...NEXT...STEP
DO...LOOP
EXIT

Chapitre 5
Calculs et tracés de courbes

Les sept opérations

Les quatre opérations ont déjà été rencontrées plusieurs fois:

+ addition
– soustraction
* multiplication
/ division

Il en existe trois autres qui sont représentées par:

^ exponentiation (à l'écran ce signe devient ↑)
@ division entière
MOD modulo

L'exponentiation, ou encore élévation à la puissance, permet d'obtenir directement le carré A^2, le cube A^3 ... la puissance n, où n est un nombre réel quelconque, de A:
Ainsi 4^3 représente 4*4*4:

```
?4^3
64
```

et 9^0.5 représente $9^{1/2}$ ou $\sqrt{9}$:

```
?9^0.5
3
```

Les deux dernières opérations (division entière et modulo) agissent sur des nombres entiers.
Lorsqu'on effectue "à la main" la division entre deux nombres et qu'on ne poursuit pas après la virgule, on obtient un quotient et un reste.

La division entière @ donne le quotient de la division entière:

```
?14@4
3
```

Le modulo (MOD) est le reste obtenu dans la division entière.

```
?14MOD4
2
```

Ordre des calculs

L'ordre dans lequel l'ordinateur effectue les calculs est le même, que les calculs portent sur des nombres ou sur des variables.
— Si le calcul à effectuer est une suite d'additions et de soustractions, ou bien une suite de multiplications et de divisions, l'ordinateur les calcule les unes à la suite des autres à partir de la gauche, comme une calculatrice.
— Si le calcul comporte à la fois des additions (ou soustractions) et des multiplications (ou divisions):

```
?3+2*5
13
```

l'ordinateur calcule d'abord toutes les multiplications (et divisions) de gauche à droite puis les additions (et soustractions).

```
?7-10/2+3
5
```

— Si le calcul comporte en outre des exponentiations, elles sont effectuées en premier. Il est donc inutile de les placer entre parenthèses:

```
3*4^2+5
  └┬┘
   16
└──┬──┘
   48
└─────┬─────┘
      53
```

```
?3*4^2+5
53
```

— Pour imposer un autre ordre que celui prévu par l'ordinateur, ce qui est assez fréquent, il faut avoir recours aux parenthèses:

```
? 12 + 4 + 8/3
18.6667

? (12 + 4 + 8)/3
8
```

La notation des nombres et leur précision

Dans l'exemple précédent nous avons eu pour premier résultat 18.6667. En fait la division ne tombe pas juste et l'ordinateur ne pouvant donner la suite infinie 18.666666666... a conservé six chiffres en arrondissant le dernier.

Six chiffres c'est la précision ordinaire de l'ordinateur. Si davantage de chiffres vous sont nécessaires, par exemple pour une comptabilité au centime près, il faut demander la "double précision". Celle-ci, en fait, fait plus que doubler puisqu'elle offre seize chiffres significatifs. Cette précision est obtenue en faisant suivre les nombres par le signe #:

```
? 12 + 4 + 8#/3#
18.66666666666667
```

et de même pour les variables:

```
X#=1234567890
? X# * 4
```

En fait tout nombre donné avec plus de sept chiffres est automatiquement considéré comme étant en double précision: aussi avons-nous donné la valeur numérique de X# sans #.

Attention: la limitation sur le nombre de chiffres (6 en simple précision, 16 en double précision) n'implique pas une limitation sur la valeur des nombres: vous pouvez utiliser des nombres compris entre 10^{38} (1 suivi de 38 zéros) et 10^{-38} ($1/10^{38}$):

```
? 123456 * 7890
? X# ^ 2
```

La réponse est exprimée en notation scientifique, comme sur les calculatrices: le chiffre qui suit E (en simple précision) ou D (en double précision) indique de combien de rangs il faut déplacer la virgule vers la droite.
7.2 E 8 représente 720 000 000.

Vous pouvez aussi utiliser cette notation pour écrire les nombres, en simple ou en double précision:

```
X#=1.23456789D9
```

Tracé de fonctions mathématiques

Un grand nombre de fonctions sont accessibles sur l'ordinateur: les fonctions usuelles sur calculatrice (sinus, cosinus, log, racine carrée...) mais aussi des fonctions plus originales comme celle qui donne la partie entière d'un nombre décimal (INT) ou le maximum d'une liste de nombres (MAX), etc. La liste complète est en page 313.

Profitons de l'écran pour tracer par exemple sinus x.
Comme pour toutes les fonctions mathématiques, la valeur de la fonction s'écrit SIN (X). L'unité est le radian.

Faisons varier X entre 0 et 10, SIN (X) variant entre −1 et 1.

A l'écran les numéros de lignes vont croissant vers le bas; l'axe vertical est donc dirigé en sens contraire de l'axe vertical usuel en mathématiques. Pour axe horizontal prenons la ligne L=100.

Par ailleurs il nous faut choisir une échelle: par exemple 30.

Les coordonnées d'un point de la courbe seront:

```
C=30*X
L= −30*SIN(X) + 100
```

Il nous reste à déterminer le pas de variation de X, ce qui se fait par tâtonnements: X variant de 0 à 10, prenons un pas de 0.1.

Le programme de tracé de la courbe sera:

```
10 'SINUS 1
20 CLS
30 FOR X=0 TO 10 STEP 0.1
40   C=30*X: L=30*SIN(X) + 100
50   PSET (C,L)
60 NEXT X
```

Pour obtenir une courbe continue, on peut tracer davantage de points en diminuant le pas de variation. Mais le tracé est beaucoup plus rapide en joignant simplement les points les uns aux autres avec une instruction LINE.
Précisons le point de départ de la courbe: X=0, soit C=0 et L=100, avant la boucle:

```
20 CLS: PSET (0,100)
```

Et modifions l'instruction de tracé en ligne 50:

```
50 LINE -(C,L)
```

Vous pouvez ainsi visualiser toute fonction usuelle: COS (X), LOG (X)...

Tracer un cercle

Un cercle peut être tracé à l'aide des fonctions SIN et COS mais il existe une instruction, CIRCLE, qui trace directement n'importe quel cercle dont on précise le centre (C,L) et le rayon R:

```
CIRCLE (200,150),30
```

dessine le cercle de centre (200,150) et de rayon 30.
Sa couleur peut être différente de celle des caractères:

```
CIRCLE (200,150),30,2
```

Le rayon ainsi que les coordonnées du centre et la couleur peuvent être définis par des variables.

Traçons par exemple une cible formée de sept cercles concentriques placés au milieu de l'écran, dans les couleurs 1, 2, 3... 7 et de rayons 10, 20, 30... 70:

```
10 'CIBLE
20 CLS: SCREEN 7,0,0
30 C=160: L=100
40 FOR COULEUR=1 TO 7
50 CIRCLE (C,L), 10 * COULEUR, COULEUR
60 NEXT COULEUR
```

La cible peut être déplacée en changeant les coordonnées du centre.

...ou une ellipse

Le cercle est prêt à tout, même à s'écraser pour se transformer en ballon de rugby, c'est-à-dire en ellipse.
Si RH et RV désignent les "rayons" horizontaux et verticaux, l'ellipse sera obtenue par:

```
CIRCLE (C,L) RH, RV
```

Attention à la syntaxe: pas de virgule avant RH; c'est cette absence de virgule qui précise que l'instruction doit dessiner une ellipse et non un cercle.

```
10 'CLS
20 CIRCLE (160,100) 70, 50
```

Et pour changer la couleur:

```
20 CIRCLE (160,100) 70, 50, 3
```

Le cercle, comme le rectangle, peut se remplir de peinture:

```
20 CIRCLEF (160,100) 70, 50, 3
```

ou d'un motif quelconque

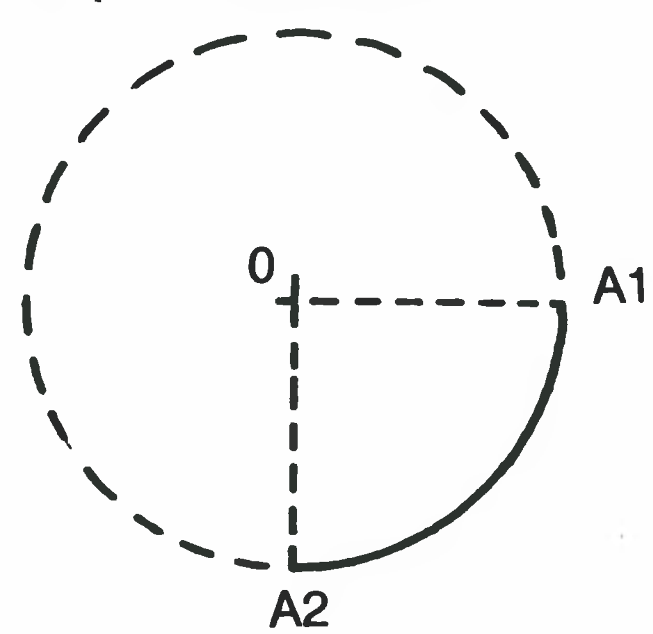
```
15 PATTERN "/"
15 PATTERN "."
```

Découper en morceaux cercles et ellipses

L'instruction CIRCLE, comme toutes les instructions graphiques, accepte de travailler avec des points en dehors de l'écran:

```
CIRCLE (160,300), 200
```

et c'est la terre vue par le hublot d'un satellite...
Mais mieux encore, vous pouvez dessiner n'importe quelle portion de cercle en n'importe quel point de l'écran.

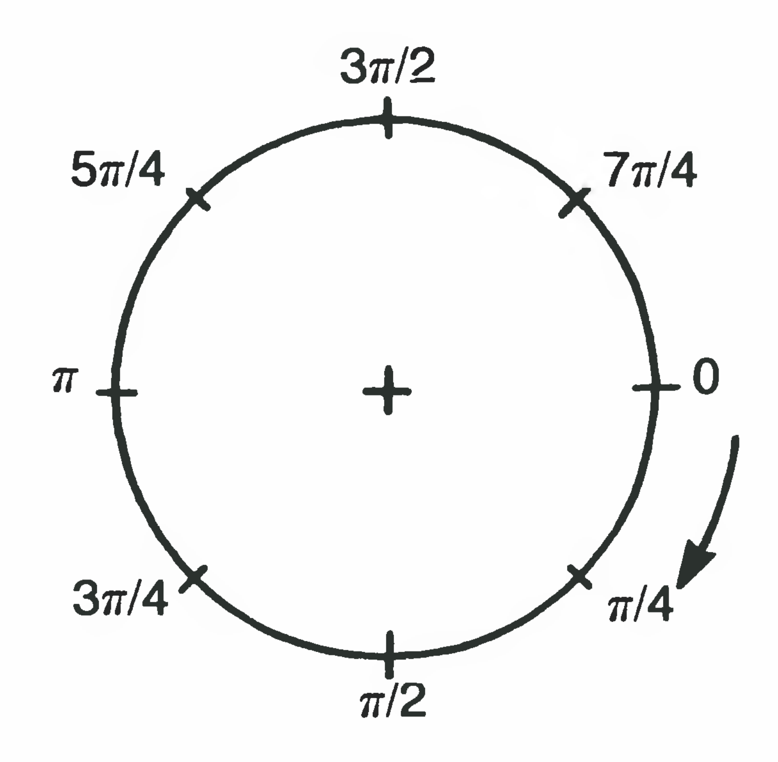


Une portion de cercle, par exemple A1 A2, est limitée par les deux rayons OA1 et OA2. La position de ces deux rayons sera précisée par les angles qu'ils font avec l'horizontale: 0° pour 0A1, 90° pour OA2.
Pour l'ordinateur, les angles doivent être exprimés en radians.

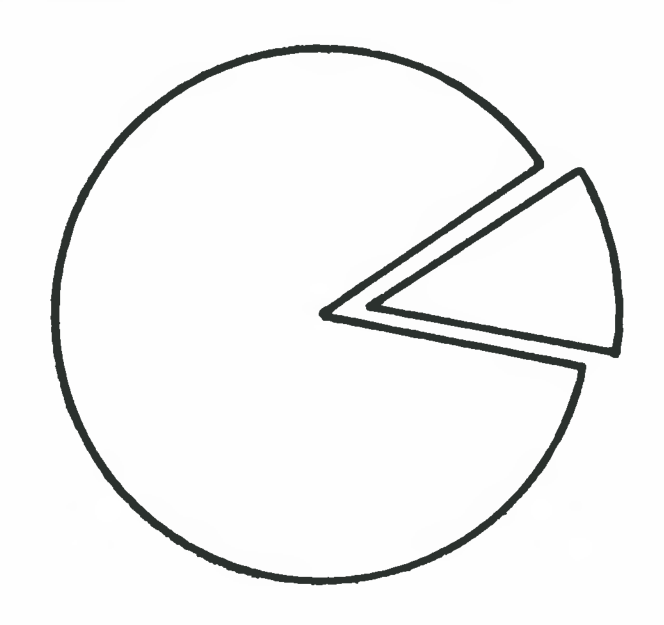
Sachant que π=3,14 radians correspondent à 180°, il ne reste plus qu'à faire une règle de 3 (ou un petit programme de conversion des degrés en radians...).
Ainsi 90° = π/2 rd= 1,57 rd
Si le cercle est placé au centre de l'écran, l'arc de cercle A1 A2 sera obtenu par:

```
CIRCLE (160,100), 50; 0, 1.57
```

Les deux angles limites sont précisés après le rayon et séparés de celui-ci par un point-virgule.
Remarquez que les angles sont comptés dans le sens des aiguilles d'une montre et à partir du point A1.
Voici les angles en radians correspondant aux 1/4 et 1/8 de cercle:



Ainsi pour faire un "camembert", c'est-à-dire illustrer une répartition en pourcentage, le cercle sera découpé en morceaux dont l'angle est proportionnel à une fraction: ici 1/8 et 7/8.



```
10 CLS: PI=3.14
20 CIRCLEF (160,100), 50; 0, 7*PI/4
30 CIRCLEF (171,96), 50, 1; 7*PI/4, 0
```

L'ellipse peut aussi être coupée en morceaux suivant la même règle, et les divers morceaux de cercle ou d'ellipse peuvent s'assembler.

Par exemple, un œuf sera formé d'un demi-cercle et d'une demi-ellipse de couleur jaune clair accolés:

```
10 CLS
20 CIRCLEF (160,100), 50, 11; 0, 3.14
30 CIRCLEF (160,100) 50, 70, 11; 3.14, 6.28
```

Il ne vous reste qu'à ajouter le coquetier.

Pour une lune à son dernier quartier — un 1/2 cercle et une 1/2 ellipse entre $\pi/2$ et $3\pi/2$ — introduisons la valeur de π dans une variable:

```
10 CLS
20 PI=3.14: SCREEN 7,0,0
30 CIRCLE (250,60), 50; PI/2, 3*PI/2
40 CIRCLE (250,60) 30, 50; PI/2, 3*PI/2
```

On pourrait ensuite construire le programme qui détermine les phases de la lune, mais ceci est une autre histoire...

Peindre une zone de couleur homogène: PAINT

Pour donner un peu de luminosité à cette lune, il faut la recouvrir d'une couleur claire.
Avec l'instruction PAINT, toute zone de l'écran de couleur homogène, c'est-à-dire tout ensemble de points contigus de même couleur, peut être coloriée.
Ajoutez la ligne suivante:

```
50 PAINT (210,60)
```

L'instruction PAINT est suivie des coordonnées d'un point situé à l'intérieur de la zone et le remplissage se fait à partir de ce point.

Si vous voulez une lune différente, rose (9) par exemple, il faut l'indiquer à la fin de l'instruction PAINT:

```
50 PAINT (210,60), 9
```

Et si un ciel noir vous paraît trop triste, essayez ceci :

```
60 PAINT (100,100), 5
```

Fonctions et instructions vues

SIN
CIRCLE
PAINT

Chapitre 6

Conserver ses programmes sur disquette ou cassette

La mémoire vive de l'ordinateur ne peut garder qu'un seul programme en tête. Et lorsqu'on éteint la machine, tout s'efface...
Les disquettes et les cassettes permettent de conserver indéfiniment des programmes et des données et de les charger dans la mémoire de l'ordinateur lorsqu'on en a besoin.
Commençons par l'utilisation des disquettes.

Formater une disquette : DSKINI

Toute disquette vierge doit subir une opération de "formatage" spécifique à chaque ordinateur.
Cette initialisation peut aussi être utilisée pour remettre à zéro une disquette dont le contenu n'est plus utile.
Entrez la disquette dans le lecteur, Fermez le volet du lecteur et tapez :

```
DSKINI 0
```

0 est le numéro du premier lecteur de disquettes (obligatoire même s'il n'y a qu'un lecteur).

Attention, le formatage détruit le contenu d'une disquette si elle n'est pas vierge. Il faut donc employer avec précaution l'instruction de formatage.

Pendant la durée du formatage, la tête de lecture balaie toute la surface de la disquette et y définit des pistes concentriques divisées chacune en secteurs égaux. La tête de lecture accède très rapidement à une piste quelconque de la disquette en se déplaçant au-dessus de la fenêtre de lecture pendant que la disquette tourne de la quantité nécessaire pour lire un secteur donné ou y écrire.

En outre le formatage organise un endroit de la disquette pour y recevoir son "répertoire", c'est-à-dire la liste de son contenu. Lorsque le message OK apparaît à l'écran, la disquette est formatée et prête à enregistrer vos programmes et vos données.
Vous pouvez donner un nom à chaque disquette lors du formatage, par exemple ESSAIS:

```
DSKINI 0,,"ESSAIS"
```

Entre les deux virgules peut être précisé un paramètre qui modifie la structure de la disquette (voir Annexe "Organisation d'une disquette").

Enregistrer un programme et le charger: SAVE, LOAD

Voici un petit programme bien sage et qui ne risque pas de remplir la disquette:

```
10 'ACCUEIL
20?BONJOUR...
```

Avant de transférer ce programme de la mémoire vive de l'ordinateur sur la disquette, vous devez d'abord choisir sous quel nom vous allez l'enregistrer. C'est sous ce nom-là qu'il sera inscrit dans le répertoire de la disquette.

Pour enregistrer le programme avec le nom ACCUEIL, tapez:

```
SAVE "ACCUEIL"
```

Un nom de programme peut contenir jusqu'à huit caractères, y compris les chiffres et signes divers (sauf le point, la virgule et les parenthèses).

L'enregistrement est terminé dès que OK reparaît à l'écran.
Si vous obtenez le message:

```
No Disk
```

c'est que le loquet n'est pas fermé, ou bien:

```
Write Protected
```

c'est que vous avez récupéré une disquette protégée contre l'écriture. Il suffit alors de retirer cette protection pour pouvoir enregistrer.
Un petit commentaire de huit caractères maximum peut être accolé au nom du programme lors de l'enregistrement:

```
SAVE "(Juliette) ACCUEIL"
```

Ce commentaire, placé entre parenthèses, n'entre pas dans le nom du programme. Vous pouvez y préciser le thème du programme (jeu...), la date, l'auteur...

Le programme étant enregistré, la mémoire de l'ordinateur peut être vidée:

```
NEW
```

Vérifiez qu'il n'y a plus rien avec LIST.
Pour recharger le programme, tapez:

```
LOAD "ACCUEIL"
```

et vérifiez le chargement:

```
LIST
```

Si vous voulez faire exécuter le programme dès qu'il est chargé, utilisez l'instruction RUN au lieu de LOAD:

```
NEW
RUN "ACCUEIL"
```

Cette ligne est équivalente à:

```
LOAD "ACCUEIL"
RUN
```

Modifier un programme enregistré

Si le programme est maintenant modifié:

```
15 CLS
20? "SALUT!"
```

et enregistré sous le même nom:

```
SAVE "(Juliette) ACCUEIL
```

la nouvelle version efface la précédente.

Exécution automatique d'un programme sur disquette: AUTO.BAT

Le chargement d'un programme n'est pas très compliqué mais il peut être rendu encore plus simple pour les utilisateurs ne connaissant pas un mot de Basic.
Un jeu pour enfants, par exemple, peut être lancé automatiquement après allumage de l'ordinateur simplement par appui sur la touche 2 du menu.

Pour cela il suffit de donner au programme le nom "AUTO.BAT".
Sauvegardons notre programme d'accueil encore en mémoire sous ce nom:

```
SAVE "AUTO.BAT"
```

Appuyez sur le bouton d'initialisation et choisissez l'option 2 dans le menu.
Vous pouvez également taper:

```
RESET
```

pour initialiser la machine, le résultat obtenu est identique.
En plus de l'annonce habituelle "Basic 128 V1.0 © Microsoft 1985", le programme affiche directement

```
SALUT!
```

Ainsi l'appui sur la touche 2 implique l'instruction suivante:

```
RUN "AUTO.BAT"
```

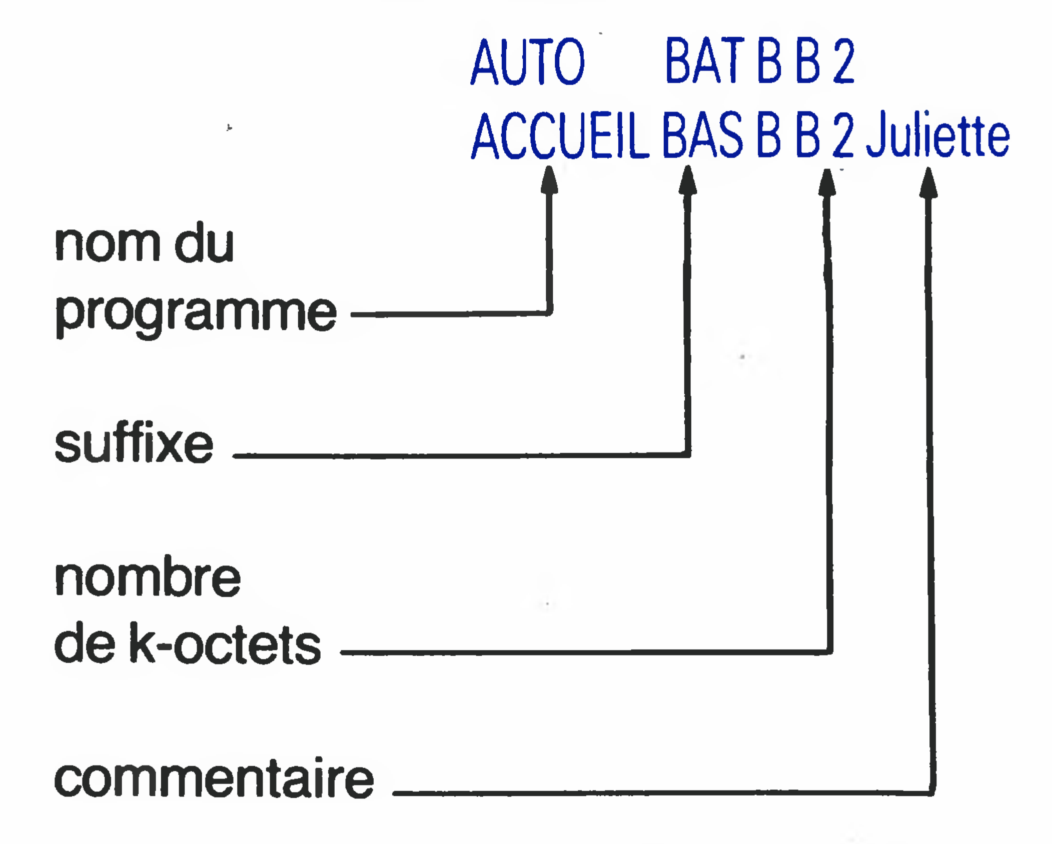
s'il existe un programme AUTO.BAT.
Cela peut être utile aussi pour détailler le contenu de la disquette, offrir un choix aux utilisateurs...

Le répertoire de la disquette: DIR

Il n'y a pas encore un grand choix sur notre disquette, mais vérifions son contenu, tapez:

```
DIR
```

La réponse est le répertoire (Directory) ou liste de tous les fichiers enregistrés sur la disquette; jusqu'à présent nous n'avons enregistré que des programmes (il existe un autre type de fichiers: les fichiers de données).
Après une ligne en vidéo inverse contenant diverses indications dont le nom que vous avez attribué à la disquette (si vous lui en avez donné un) vous voyez:



Le répertoire sait aussi être sélectif:
— si vous ne voulez voir que la liste des programmes dont le nom commence par la lettre T, demandez

DIR "T"

pour les programmes dont le nom commence par "DESSIN" :

DIR "DESSIN"

— si vous ne voulez lister que les programmes dont le suffixe est

DIR ".MUS"

Le répertoire peut aussi être consulté sur papier, si vous possédez une imprimante:

DIRP

commandera l'impression du répertoire.

Fusionner deux programmes: MERGE

L'action réalisée par un programme: l'affichage d'un texte à l'écran, le jeu d'un indicatif musical, le dessin d'une certaine figure, un calcul de TVA ... peuvent être utiles dans des programmes différents.

L'instruction MERGE (fusionner) permet de charger un programme enregistré sur disquette dans la mémoire centrale sans effacer le programme qui s'y trouve (le programme "résident").

Mais le programme sur disquette doit avoir été sauvegardé d'une manière particulière, en ASCII. Ceci est le nom du code américain, devenu international, faisant correspondre un nombre à chaque caractère. Pour sauvegarder un programme en ASCII, par exemple le programme "ACCUEIL", tapez:

```
SAVE "ACCUEIL", A
```

Demandez maintenant le répertoire:

```
DIR
```

Le programme se trouve cette fois en ASCII (A dans la quatrième colonne).

Nous pouvons maintenant entrer un autre programme en mémoire, qui ne contienne pas de numéros de lignes communs avec le programme à fusionner:

```
100? "VOICI UNE ELLIPSE"
110 CIRCLEF (160,100) 50,70,5
```

Et maintenant, chargeons le programme "ACCUEIL" en mémoire en le fusionnant au programme résident:

```
MERGE "ACCUEIL"
LIST

10 ' ACCUEIL
15 CLS
20? "SALUT!"
100? "VOICI UNE ELLIPSE"
110 CIRCLEF (160,100) 50,70,5
```

Les deux programmes sont bien enchaînés et vous pouvez sauvegarder l'ensemble:

```
SAVE "ELLIPSE"
```

Tenir secrets ses programmes

Si vous voulez conserver secrets certains de vos programmes, par exemple ELLIPSE, sauvegardez-les sous forme “protégée” :

```
SAVE "ELLIPSE",P
```

Sous cette forme, le programme peut être chargé et exécuté, mais non listé; la suite des instructions n’est plus accessible.
Attention: si vous n’avez pas gardé en lieu sûr (sur un papier... ou sur une autre disquette) une copie non protégée de votre programme, vous ne pourrez plus jamais le lister ni le modifier.

Changer le nom d’un programme: NAME

Il est souvent intéressant de conserver au moins temporairement des versions successives d’un même programme.

Si vous avez fait une deuxième version du programme ELLIPSE et que vous désiriez conserver la première, vous pouvez rebaptiser la première version ELLIPSE1 :

```
NAME "ELLIPSE.BAS" AS "(ancienne)ELLIPSE1.BAS"
```

Le suffixe doit être précisé dans l’instruction NAME et il est possible de mettre un commentaire avec le nom du fichier.
Vérifiez le résultat de l’opération:

```
DIR
```

Si par mégarde vous proposez comme nouveau nom le nom d’un programme qui existe déjà sur la disquette, vous êtes prévenu du risque d’écraser ce programme par:

```
Already Exists     (Existe déjà...)
```

Effacer un programme de la disquette: KILL

Lorsque vous êtes satisfait de votre dernière version, il ne vous reste plus qu’à jeter au panier les ébauches successives. C’est facile si vos programmes sont numérotés (ELLIPSE1, ELLIPSE2...) ou si leur date de réalisation est précisée dans le commentaire du répertoire.

Pour effacer un programme de la disquette et récupérer la place correspondante, il suffit de donner l'ordre brutal:

```
KILL "ELLIPSE1.BAS"
```

Comme avec l'instruction NAME, il est indispensable de préciser le suffixe, c'est-à-dire BAS si vous n'en avez pas donné.

Vérifions le résultat du crime:

```
DIR
```

Le programme ELLIPSE1.BAS a disparu.

Utilisation des cassettes

Si vous utilisez le lecteur-enregistreur de programmes, les choses se simplifient.

Pour sauvegarder un programme, il faut d'abord se placer à l'endroit voulu de la cassette, c'est-à-dire ni au début (à l'endroit de la bande amorce), ni au même endroit qu'un autre programme déjà enregistré. Le positionnement de la bande se fait avec les touches de défilement rapide avant et arrière qui sont toujours actives. Puis il faut appuyer simultanément sur la touche d'enregistrement et la touche de lecture. La cassette est bloquée.

Il faut alors donner l'ordre de sauvegarde:
SAVE "CASS:ACCUEIL"

Le magnétophone se met en route et au bout de quelques secondes, l'enregistrement est terminé et le message "OK" reparaît.

La lecture du programme s'effectue alors de la manière suivante:
— Rembobinez la cassette jusqu'au début du programme. Il est toujours bon de noter le numéro du compteur au moment de la sauvegarde. Sinon, rembobinez jusqu'au début de la cassette.
— Appuyez sur la touche de lecture du magnétophone.
— Tapez:

```
LOAD "CASS:ACCUEIL"
```

Le moteur se met en marche et à l'écran s'affiche:

```
Searching
```

puis, lorsque le début du programme est trouvé:

```
Found: ACCUEIL.BAS
```

et à la fin du chargement:

```
OK
```

Il n'y a plus qu'à l'exécuter.

Si vous ne mettez qu'un seul programme par cassette, il n'est plus nécessaire d'indiquer le nom du programme au moment du chargement. Il suffit de faire:

```
LOAD"CASS:"
```

C'est le premier programme présent sur la cassette qui est alors chargé.

Instructions vues

DSKINI	MERGE
SAVE	NAME
LOAD	KILL
DIR	RESET

Chapitre 7
Chaines de caractères

Le langage Basic manipule des nombres et des textes. Tout ce qui ne doit pas servir à des calculs est considéré comme du texte. Cela constitue des “chaînes de caractères”. On les écrit toujours entre deux guillemets:

"BONJOUR"
"4 AOUT 1789"
"333-33-33"

Une chaîne de caractères peut contenir des lettres, des chiffres, des signes de ponctuation, des blancs, ... bref tous les caractères disponibles sur votre micro-ordinateur.
La longueur d'une chaîne peut dépasser une ligne de l'écran mais elle est limitée à 255 caractères (plus de six lignes).

Variables

Pour conserver des chaînes en mémoire, par exemple une liste de livres ou d'adresses, et ensuite les réutiliser, il va falloir les mettre dans des variables différentes.
C'est le signe $ à la fin du nom d'une variable qui indique que son contenu est une chaîne de caractères.
A$, N1$, REPONSE$ sont des noms possibles de chaînes de caractères:

```
NOM1$="DUPONT"
NOM2$="DUPOND"
PRINT NOM1$;" ET ";NOM2$
```

On ne peut pas confondre une variable chaîne avec une variable numérique puisque la première se termine par un $:

```
A=123
A$="STAR WARS"
```

définissent deux variables A et A$ de types différents.
Mais il est interdit d'affecter une valeur numérique à une variable chaîne de caractères:

```
A$=144/12
Type Mismatch
```

L'ordinateur n'aime pas les mélanges de types et il vous le dit de cette manière!
Par contre vous pouvez toujours définir la variable A$="144/22" comme la suite des caractères 1,4,4,/,2,2.

Fabriquer de nouvelles chaînes de caractères à partir de celles qui existent se fait par trois modes de transformation différents:
— mettre deux chaînes bout à bout: concaténation,
— extraire un morceau d'une chaîne,
— remplacer un morceau d'une chaîne par un autre.

Concaténation: les chaînes bout à bout

La concaténation (qui vient du latin *catena*: chaîne) consiste à enchaîner une chaîne à la suite d'une autre pour en faire une nouvelle. Elle est notée par le signe +:

```
A$="CHRISTOPHE"
B$="COLOMB"
? A$+B$
CHRISTOPHECOLOMB
```

Pour fabriquer des phrases à partir de mots, il est préférable de mettre des espaces pour les séparer:

```
? A$+" "+B$
```

Bien pratique pour des exercices d'écriture automatique, mais attention à ne pas dépasser 255 caractères, sinon vous aurez droit à:

```
String Too Long      (Chaîne trop longue)
```

Extraction: la chaîne en morceaux

Nous allons voir comment couper une chaîne en morceaux, sur les bords ou au milieu. Toutes ces extractions se font avec des fonctions dont le nom se termine par un $.

Prenons une chaîne de caractères:

```
A$="AMERIQUE"
```

Comment prendre une partie de la chaîne?

— à partir de la gauche: fonction LEFT$
LEFT$(A$,4) a pour résultat AMER
LEFT$(A$,1) a pour résultat A

Le premier argument de la fonction - A$ - indique la chaîne qui sert à l'extraction, le deuxième argument - 4 ou 1 - indique le nombre de caractères à prendre.

La fonction peut être appliquée directement sur une chaîne et non sur une variable:

```
? LEFT$ ("DUMOULIN",2)
DU
```

— à partir de la droite: fonction RIGHT$
RIGHT$ (A$,1) a pour résultat E
RIGHT$ (A$,3) a pour résultat QUE

Le premier argument indique la chaîne qui sert à l'extraction et le second le nombre de caractères à extraire.
RIGHT$ et LEFT$ ont un effet symétrique.

— au milieu: fonction MID$ (de *middle*: milieu)
pour extraire un caractère à partir du quatrième, puis du deuxième:
MID$ (A$,4,1) a pour résultat R
MID$ (A$,2,1) a pour résultat M

pour extraire 3 caractères à partir du deuxième, puis 4 à partir du premier:
MID$ (A$,2,3) a pour résultat MER
MID$ (A$,1,4) a pour résultat AMER

pour extraire jusqu'à la fin de la chaîne, à partir du sixième caractère:
MID$ (A$,6) a pour résultat QUE

La fonction MID$ a trois arguments:
— le premier indique la chaîne qui sert à l'extraction,
— le deuxième indique la position du premier caractère à extraire,
— le troisième indique le nombre de caractères à extraire, il n'est pas obligatoire.

A quoi cela peut-il servir?
Problème: afficher une chaîne de caractères verticalement.
Le premier caractère sur la première ligne, le deuxième sur la deuxième ligne... etc.
L'action à répéter est la suivante: extraire le caractère correspondant au numéro de la ligne et l'afficher.
Le nombre de répétitions est connu: il est égal au nombre de caractères de la chaîne. On fera donc une boucle FOR... NEXT:

```
10 'ECRITURE VERTICALE
20 CLS
30 A$="ABCDEFGHIJKLMNOPQRSTUVW"
40 FOR I=1 TO 23
50 PRINT MID$ (A$,I,1)
60 NEXT I
```

La chaîne A$ fait exactement 23 caractères; tout tombe juste. Pour afficher verticalement une autre chaîne, il faut alors modifier la valeur de fin de boucle en ligne 40.

Pour ne pas avoir à modifier les deux lignes, nous allons utiliser une fonction qui donne le nombre de caractères d'une chaîne: c'est la

fonction LEN (*length*=longueur). Remarquez que son nom ne se termine pas par un $: le résultat qu'elle fournit n'est pas une chaîne de caractères mais un nombre.
La ligne 40 devient:

```
40 FOR I=1 TO LEN (A$)
```

Vous pouvez maintenant mettre la chaîne que vous voulez en ligne 30 dans A$.

Autre problème: inverser une chaîne de caractères.
Le premier caractère devient le dernier, le second l'avant-dernier et ainsi de suite. C'est un premier pas vers le verlan...
Comme dans l'exemple précédent, il s'agit d'extraire un caractère après l'autre et de l'afficher; mais cette fois en commençant par la fin.

```
10 'INVERSE
20 A$="LAVAL"
30 FOR I=LEN (A$) TO 1 STEP-1
40 PRINT MID$ (A$,I,1);
50 NEXT I
```

Ce petit programme va vous aider à rechercher les mots qui se transforment en un autre mot quand on les lit à l'envers ou bien ceux qui restent inchangés.

Entrer des chaînes: INPUT, LINEINPUT

Dans le programme précédent, il serait agréable de ne pas avoir à changer la ligne 20 à chaque essai.
Le plus simple est de demander à l'utilisateur le nom qu'il veut inverser et d'introduire la réponse dans A$ avec INPUT:

```
20 INPUT "Quel mot";A$
RUN
```

A l'exécution de la ligne 20, c'est-à-dire pratiquement aussitôt, un point d'interrogation s'affiche après la question.
Il suffit de répondre par un mot, ou plus, sans oublier de valider la réponse avec ENTREE.
Attention à ne pas mettre de virgule dans la réponse, vous auriez droit à un message désagréable du genre:
"Redo from start" (Recommencez au début)

C'est compréhensible : l'interpréteur Basic considère que les réponses à l'instruction INPUT sont séparées par des virgules. Donc s'il y a une virgule dans la réponse, il croit qu'il y a deux réponses et comme il n'en attendait qu'une...
Si vous tenez vraiment à la virgule, vous pourrez la garder mais il faudra mettre votre réponse entre deux guillemets.

Si vous souhaitez que la réponse puisse contenir vraiment n'importe quoi, il faut utiliser l'instruction LINE INPUT.

```
20 LINE INPUT "Quel mot ";A$
```

Cette nouvelle possibilité va permettre d'améliorer le programme en acceptant les virgules dans votre réponse.

Remplacement : changer les maillons de la chaîne

Les fonctions que nous avons étudiées : LEFT$, RIGHT$, MID$ et la concaténation donnent comme résultat une nouvelle chaîne.
Avec ces fonctions on peut découper une chaîne et en faire une nouvelle avec des morceaux de l'ancienne, d'autres morceaux... La chaîne primitive est conservée.

Parfois on veut simplement remplacer dans une chaîne un certain nombre de caractères par un nombre égal d'autres caractères.
L'instruction qui réalise cela s'appelle MID$, comme la fonction, mais elle s'emploie différemment.

Essayons :

```
F$="LE GRAND SERGE"
MID$ (F$,4,5)="PETIT"
? F$
LE PETIT SERGE
MID$ (F$,10,5)="HOMME"
? F$
LE PETIT HOMME
```

MID$ (F$,10,5)="HOMME" pourrait se lire de la façon suivante : remplacer, dans la chaîne F$, le morceau commençant au dixième caractère, de longueur cinq caractères, par la chaîne "HOMME".

La syntaxe de cette instruction est un peu particulière: on met l'instruction à gauche, avec comme arguments la chaîne à transformer, la position du premier caractère à remplacer, le nombre de caractères à remplacer, puis le signe =, puis la chaîne de substitution à droite.

Un peu compliqué certes, mais si pratique!
Pour en voir l'intérêt, faites la même chose sans utiliser l'instruction MID$.

Tester une réponse

Supposons que vous veniez de faire un programme passionnant: par exemple un calcul d'impôts. Ce programme est tellement intéressant qu'à la fin de l'exécution vous souhaitez peut-être le recommencer.

Il suffit alors de poser la question "Voulez-vous recommencer?" et de tester si la réponse est OUI ou NON.
Nous n'étudierons ici que la manière d'analyser la réponse à la fin du programme.

```
10 ' PROGRAMME INTERESSANT
. . .
100 INPUT "VOULEZ-VOUS RECOMMENCER";REP$
110 IF REP$="OUI" THEN 10
```

Mettez entre la ligne 10 et la ligne 100 toutes les instructions qui vous plaisent.

Le test de la réponse est réduit au minimum:
si la réponse est égale à OUI, alors aller au début, sinon (sous-entendu) continuer à l'instruction suivante donc ici terminer.

La comparaison des chaînes de caractères se fait, comme pour les nombres, avec le signe =.
Deux chaînes sont égales si elles ont le même nombre de caractères et si tous leurs caractères sont identiques.

Ajoutons, pour plus de politesse, un message pour l'utilisateur qui répond NON:

```
120 IF REP$="NON" THEN PRINT "AU REVOIR"
```

Que faire maintenant des indécis, ceux qui n'ont répondu ni OUI, ni NON; il faut les amener à se décider:

```
120 IF REP$="NON" THEN PRINT "AU REVOIR" ELSE 100
```

Cette ligne de programme donne un exemple complet de la syntaxe de IF... THEN... ELSE en trois parties:
— entre IF et THEN: la condition à tester
— entre THEN et ELSE: ce qu'il faut faire si le résultat est VRAI
— à partir de ELSE jusqu'à la fin de la ligne: ce qu'il faut faire si le résultat est FAUX.

Il est possible de mettre plusieurs instructions entre THEN et ELSE, ou après ELSE.

```
120 IF REP$="NON" THEN CLS: LOCATE 16,10: PRINT "AU REVOIR" ELSE PRINT
"FAUDRAIT SAVOIR!" : GOTO 100
```

On peut encore améliorer l'efficacité du programme, en ne testant que la première lettre de la réponse avec LEFT$:

```
110 IF LEFT$ (REP$,1)="O" THEN 10
```

Ainsi ceux qui tapent O, OU ou OUI, toujours suivi de ENTREE, sont rangés dans les OUI. Reste à faire la même modification pour tester NON en ligne 120.

Tester une réponse d'un caractère: INPUT$

Dans de nombreux logiciels, il suffit de taper O pour OUI, N pour NON, S pour SUITE... etc., sans avoir à valider sa réponse par ENTREE.

La fonction INPUT$ va nous le permettre.
Attention, il ne s'agit pas d'une instruction comme INPUT mais d'une fonction comme RIGHT$; elle rend donc un résultat qu'il faut affecter à une variable, ou traiter directement. On indique dans INPUT$ le nombre de caractères attendus, et la fonction retourne les caractères tapés au clavier. Le test O ou N peut alors se réécrire:

```
100 PRINT "VOULEZ VOUS RECOMMENCER?" ;
110 REP$=INPUT$ (1)
120 IF REP$="O" THEN 10
130 IF REP$="N" THEN PRINT "AU REVOIR" ELSE 100
```

Après avoir posé la question avec PRINT, en n'oubliant pas le point d'interrogation car INPUT$ ne l'affiche plus, l'interpréteur Basic attend un caractère. Dès que celui-ci est tapé, il l'affecte à REP$, et le programme se poursuit.

Remarque: le caractère tapé au clavier n'apparaît pas.
Pour le voir apparaître, il faut l'afficher:

```
110 REP$=INPUT$(1): PRINT REP$
```

La possibilité de tester une réponse de plusieurs caractères peut servir à mettre des mots de passe à l'entrée de vos logiciels, pour éviter les usages indiscrets. Voici comment placer le test dans les premières instructions:

```
20 PRINT "MOT DE PASSE"
30 REP$=INPUT$(7)
40 IF REP$ <> "ZORGLUB" THEN NEW
```

En ligne 30, l'interpréteur ne passe à la suite que si l'utilisateur a tapé sept caractères.
Pour tester la différence de deux chaînes, on utilise les signes <>, comme pour les nombres.
En ligne 40, l'instruction NEW supprime tout ce qui est présent en mémoire!
Ainsi le programme ne peut pas être exécuté sans connaissance du mot de passe. Pour que ce sésame en soit vraiment un, il faut que l'utilisateur ne puisse lister le programme et donc que celui-ci soit sauvegardé sous forme protégée.

Fonctions et instructions vues

LEFT$	LEN
RIGHT$	LINE INPUT
MID$	LINE INPUT$

Chapitre 8
Le hasard, la logique et le crayon optique

Produire des nombres au hasard : RND

L'ordinateur ne fait que ce que lui demande le programme. Donc le même programme produit toujours les mêmes effets, pour les mêmes valeurs des variables.
Comment faire varier automatiquement les effets d'un programme?
En introduisant le hasard.

La fonction qui produit les nombres au hasard s'appelle RND.

```
? RND
```

L'ordinateur affiche :

```
.594972
```

Recommençons :

```
? RND
.512679
```

Le nombre obtenu a changé.

En continuant ainsi, on obtient des nombres, tous différents et tous compris entre 0 et 1.
Pour bien vérifier cette assertion, écrivez une boucle infinie:

```
DO: PRINT RND: LOOP
```

Les nombres défilent, tous inférieurs à 1, de valeurs très différentes. En particulier, les plus petits sont affichés en notation scientifique avec E−2 ou E−3. On obtient ainsi une suite de nombres aléatoires.

Le hasard dans les jeux: RND et INT

Comment utiliser cette fonction pour remplacer un dé dans un jeu? Il nous faut tirer au hasard un nombre compris entre 1 et 6. On peut l'obtenir en multipliant le résultat de RND par 6.

```
DO:? 6*RND: LOOP
```

La question maintenant est de supprimer les décimales.

La fonction INT (*integer*: entier) a pour résultat la partie entière d'un nombre:

```
? INT (2.5)
2
? INT (.72)
0
```

Essayons-la sur les nombres aléatoires:

```
DO:? INT (6*RND): LOOP
```

Pour obtenir le bon résultat, c'est-à-dire un nombre compris entre 1 et 6, il faut encore ajouter 1:

```
DO:? INT (6*RND)+1: LOOP
```

Voici le programme qui permet de tirer les résultats du loto chaque semaine:

```
10 ' LOTO
100 PRINT "NUMEROS GAGNANTS" ;
110 FOR I=1 TO 6
120 PRINT INT (49*RND)+1
130 NEXT I
140 PRINT "NUMERO COMPLEMENTAIRE" ; INT (49*RND)+1
```

Quand on recommence l'exécution du programme, il donne bien sept numéros différents mais ce sont toujours les mêmes....

En effet, la commande RUN initialise toujours la position de départ de la fonction RND à la même valeur.
Ceci peut être un avantage quand on élabore un programme: c'est toujours la même série qui est essayée.
Mais c'est un inconvénient majeur pour tirer les résultats du loto!

Pour varier le hasard, lire le clavier au vol: INKEY$

Pour faire varier les sept nombres nécessaires, il faut pouvoir les prendre au hasard dans la succession des nombres fournis par la fonction RND.

La solution consiste à faire défiler ces nombres et à s'arrêter au hasard sur l'un d'eux qu'on prendra comme premier nombre pour notre tirage.

Pour arrêter le défilement, nous allons utiliser une fonction qui permet de savoir si une touche du clavier a été appuyée: INKEY$.
Le résultat de cette fonction est:
— le caractère correspondant à la touche appuyée, si une touche est appuyée,
— une chaîne vide si aucune touche n'a été appuyée.

Avant le tirage des numéros, il faut ajouter:

```
20 Z=RND: IF INKEY$="" THEN 20
```

A chaque exécution de la boucle, la variable fantôme Z prend une nouvelle valeur aléatoire. La boucle ne se termine que lorsque l'on tape sur une touche quelconque du clavier.

Comme la boucle s'exécute très vite, il est impossible de savoir combien d'appels à la fonction RND ont été faits quand on appuie sur le clavier. Les numéros sont alors vraiment tirés au hasard.

Graphiques aléatoires

Il n'est pas toujours facile de se rendre compte de l'effet du hasard. La fonction aléatoire donne-t-elle des valeurs parfaitement réparties dans l'intervalle (0,1)?

Nous pouvons le vérifier par le petit programme suivant: sur fond sombre, on affiche des points dont les coordonnées sont tirées au hasard.

```
10 REM PLUIE ALEATOIRE
20 SCREEN 7,0,0: CLS
30 DO
40    COL=INT(320*RND)
50    LIG=INT(200*RND)
60    PSET(COL,LIG)
70 LOOP
```

Petit à petit, tout l'écran se remplit. Aucune région n'est épargnée. C'est la preuve que la distribution se fait bien uniformément.

On peut faire un peu plus joli en tirant aussi la couleur des points au hasard:

```
55 C=INT(16*RND)
60 PSET(COL,LIG),C
```

Vous pouvez inventer un grand nombre d'images en utilisant le hasard: tracer des lignes, des boîtes, des cercles au hasard, en imposant certaines contraintes, etc.

Simulation graphique

Le hasard est d'une grande utilité pour simuler des phénomènes, c'est-à-dire les reproduire en explorant toutes les variations des paramètres.

A titre d'exemple, imaginons une particule, ou un objet, qui se déplace sur une zone fermée (l'écran). Au cours de ses déplacements, il se heurte à d'autres objets ou aux parois, ce qui l'amène à changer de direction. Comme il y a beaucoup de monde, il ne peut pas aller trop loin.

Un modèle de comportement consiste à tirer au hasard le déplacement (en ligne et en colonne) et à en limiter la longueur. En faisant l'hypothèse que les composantes en lignes et en colonnes du déplacement de l'objet sont des nombres au hasard

compris entre 0 et K/2, on peut écrire ces composantes, DC pour les colonnes et DL pour les lignes:

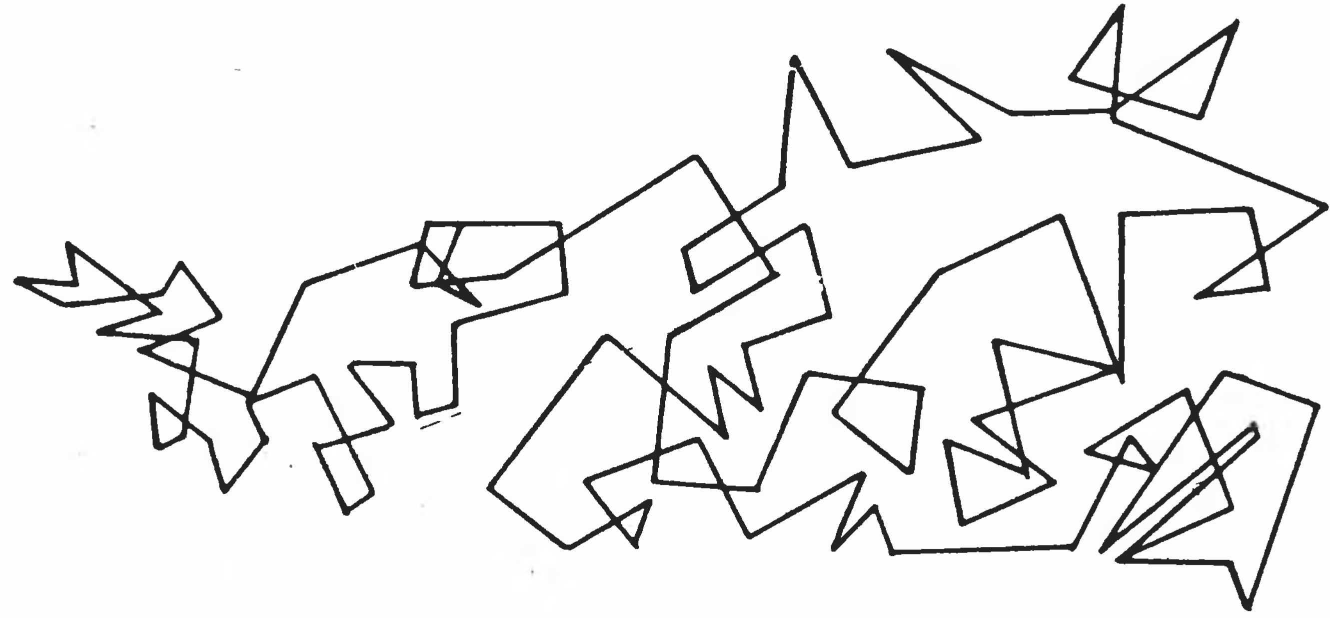
```
DC=K*(RND-0.5)
DL=K*(RND-0.5)
```

La structure générale du programme est la suivante:
— effacer l'écran,
— placer le point de départ au centre,
— répéter les actions suivantes:
 * calculer un déplacement,
 * calculer les nouvelles valeurs de colonne et de ligne
 * tracer le segment qui va au nouveau point.

En voici le listage:

```
10 ' PARTICULE
20 CLS
30 C=160: L=100: PSET (C,L)
40 K=30
50 '
100 DO
110   DC=K*(RND-.5): DL=K*(RND-.5)
120   C=C+DC: L=L+DL
130   LINE-(C,L)
140 LOOP
```

A l'exécution, on voit la trace d'une particule qui s'agite dans tous les sens.



Malheureusement, elle a tendance à s'échapper de l'écran, à y revenir, puis à en repartir.
Essayons de la maintenir dans les limites de l'écran: si le nouveau numéro de ligne dépasse 199 ou devient négatif, alors le déplacement n'est pas possible.

Comme seul le hasard nous guide, si le déplacement mène en dehors de l'écran, il faut en calculer un autre.
Ceci se traduit en Basic:

```
112 IF L+DL>199 THEN 110
114 IF L+DL<0 THEN 110
```

Pour être complet, il faut tester un débordement éventuel en colonne:

```
116 IF C+DC>319 THEN 110
118 IF C+DC<0 THEN 110
```

Cette fois, la particule est enfermée dans sa boîte.

Combiner des conditions: les opérateurs logiques

Les lignes 112 à 118 du programme précédent explicitent quatre conditions différentes qui conduisent à la même action.

Il est possible de regrouper plusieurs conditions dans la même instruction de test IF... THEN, au moyen des opérateurs logiques.

Dans le cas présent, il suffit que l'une des conditions soit remplie pour que l'action (aller en 110) soit à faire. L'opérateur logique à utiliser est OR (OU logique).

On peut alors réécrire les quatre lignes en une seule:

```
112 IF L+DL>199 OR L+DL<0 OR C+DC>319 OR C+DC<0 THEN 110
```

et supprimer les lignes 114, 116 et 118.

L'instruction de test peut se lire de la façon suivante:
— s'il y a débordement, alors aller recalculer le déplacement. En fait, on peut envisager directement la proposition contraire:
— s'il n'y a pas de débordement, alors calculer les nouvelles coordonnées et tracer le segment qui y arrive.

La négation d'une condition s'écrit avec NOT.
Les lignes 112, 120 et 130 sont remplacées par une seule ligne:

```
120 IF NOT (L+DL>199 OR L+DL<0 OR C+DC>319 OR C+DC<0) THEN C=C+DC:
L=L+DL: LINE-(C,L)
```

C'est un peu moins lisible mais plus compact et tout aussi clair.

Reste une petite imperfection: pourquoi s'exprimer de manière négative?
Essayons de supprimer NOT.
S'il n'y a pas de débordement, c'est que les nouvelles coordonnées restent dans les limites de l'écran, et donc que la nouvelle ligne est inférieure ou égale à 199, supérieure ou égale à 0, et que la nouvelle colonne est inférieure ou égale à 319 et supérieure ou égale à 0.
La relation supérieur ou égal se note >=.
La relation inférieur ou égal se note <=.
L'opérateur logique à utiliser est alors AND.
Voici donc la ligne réécrite:

```
120 IF L+DL<=199 AND L+DL>=0 AND C+DC<=319 AND C+DC>=0 THEN
L=L+DL: C=C+DC: LINE-(C,L)
```

Tout ce qu'il faut pour faire des comparaisons

Lorsque vous avez deux nombres à comparer, voici les opérateurs utiles:
= égal à
<> différent de
> supérieur à (mais pas égal)
< inférieur à (mais pas égal)
>= supérieur ou égal à
<= inférieur ou égal à

Vous pouvez écrire l'opposé d'une condition avec NOT.

Vous pouvez regrouper plusieurs conditions avec AND (ET), OR (OU) et XOR (OU exclusif). Ce dernier opérateur signifie "l'un ou l'autre, mais pas les deux à la fois". Il est d'usage moins fréquent que les deux premiers.

Le crayon optique

Le crayon optique est à la fois un instrument de communication avec les logiciels et un instrument de dessin.

Si vous n'avez pas encore effectué le réglage du crayon optique, appuyez sur le bouton d'initialisation pour revenir au menu initial, ou tapez RESET au clavier et faites ce réglage.
Lorsque vous visez un point de l'écran avec le crayon, l'ordinateur connaît en permanence les numéros de colonne et de ligne de ce point.

Plusieurs instructions de Basic permettent de les connaître également.
L'instruction INPEN retourne dans deux variables les coordonnées du point visé, même si le crayon optique n'appuie pas directement sur l'écran.

Voici un petit programme qui permet de connaître les coordonnées du point visé:

```
10 ' POINT VISE
20 CLS
30 DO
40 INPEN C,L
50 LOCATE 0,0: PRINT "COLONNE" ; C, "LIGNE" ; L
60 LOOP
```

Dès que vous avez lancé l'exécution du programme, vous voyez s'afficher les coordonnées −1 et −1. Puis, en approchant le crayon optique, les nombres défilent rapidement en face de COLONNE et de LIGNE.
Vous pouvez vérifier que le numéro de colonne varie bien de 0 à gauche à 319 à droite (il n'est même pas très facile de viser les points du bord!) et que le numéro de ligne varie de 0 en haut à 199 en bas.

Si les coordonnées s'obstinent à conserver les valeurs −1,
— ou bien vous visez obstinément en dehors de la zone utile de l'écran,
— ou bien la couleur de fond est trop foncée (noir ou rouge),
— ou bien votre téléviseur n'envoie pas assez de lumière au crayon optique. Il faut dans ce cas augmenter la luminosité du téléviseur.

Pour visualiser le point visé, il suffit de l'afficher avec l'instruction PSET:

```
55 PSET (C,L)
```

L'écran se remplit alors de petits points correspondant à chaque endroit que vous avez pointé.

Quand le résultat de INPEN est – 1 et – 1, aucun point n'est allumé. Pour afficher uniquement des points de coordonnées déterminées, et non pas tous les points visés, utilisons la fonction PTRIG .
Cette fonction indique si le crayon optique est appuyé sur l'écran ou non: modifions la ligne 55

```
55 IF PTRIG THEN PSET (C,L)
```

Cette ligne se lit: si le crayon optique est appuyé alors afficher le point C,L.

Seuls sont alors dessinés à l'écran les points visés lors de la pression sur le téléviseur.

Le même résultat peut être obtenu en une seule instruction: INPUTPEN; voyez-en l'effet avec le programme suivant:

```
30 DO
40   INPUTPEN C,L
50   LOCATE 0,0: PRINT "COLONNE" ; C, "LIGNE", L
55   PSET (C,L)
60 LOOP
```

Fonctions et instructions vues

RND	OR, AND, NOT
INT	INPEN
INKEY$	PTRIG

Chapitre 9
Un peu de musique

Le haut-parleur du téléviseur sort habituellement d'autres sons que le "bip" sonore correspondant à l'appui d'une touche. A défaut de synthétiser la parole d'une speakerine, il pourra jouer de la musique.

Les notes

L'instrument obéit à l'ordre PLAY (joue) suivi du nom des notes à jouer, la note *sol* étant raccourcie en SO:

```
PLAY "DOREMIFASOLASI"
```

Cinq octaves sont accessibles numérotées de 1 pour la plus grave à 5 pour la plus aiguë. Lorsque rien n'est précisé, comme dans l'instruction précédente, l'instrument joue automatiquement à l'octave supérieure 4.

Pour jouer la gamme à l'octave inférieure, n° 3:

```
PLAY "O3DOREMIFASOLASI"
```

Les deux caractères O3 agissent sur toutes les notes suivantes. Pour revenir à l'octave 4, il faut le préciser:

```
PLAY "O3DOREMIFASOLASIO4DO"
```

La pause, représentée par P, reste seule indifférente à l'octave... les dièses et les bémols se précisent par # et b après le nom de la note à modifier:

PLAY "DO# MIb"

Le rythme

La durée des notes peut varier à volonté sur une échelle comprise entre 0 et 96, la durée standard correspondant au nombre 24.
Pour modifier cette durée, on la précise devant la note:

PLAY "L96DOL48REMIL24FA"

Comme pour l'octave, les caractères "L48" affectent toutes les notes qui les suivent jusqu'à la nouvelle durée. Pour revenir à la durée standard il faut préciser L24.

Ainsi:
L96 correspond à une ronde
L48 correspond à une blanche
L24 correspond à une noire
L12 correspond à une croche
L6 correspond à une double croche

Si vous voulez jouer plus subtil, vous pourrez aussi modifier le tempo et l'amortissement des notes... (voir Référence)

Pianoter à l'écran: ON PEN GOTO...

Pour ce piano, les touches seront des cases à l'écran et la désignation d'une touche avec le crayon optique déclenchera le jeu de la note correspondante.

Le couple d'instructions PEN et ON PEN...GOTO permet:
— de définir jusqu'à huit cases différentes à l'écran (ici sept nous suffiront),
— de demander l'exécution d'une ligne différente pour chaque case lorsqu'une case est désignée au crayon optique. C'est exactement ce qu'il nous faut.

Quelle sera la structure du programme?

• Affichons d'abord le nom des notes, espacées d'un caractère :

```
10 ' GAMME
20 CLS: SCREEN 0,6,6
30 LOCATE 3,11,0
40 PRINT "DO RE MI FA SO LA SI"
```

Le chiffre 0 en troisième position dans l'instruction LOCATE rend le curseur invisible pendant le déroulement du programme.

• Définissons sept cases de trois caractères en largeur (24 colonnes) et deux caractères en hauteur (16 lignes), entourant les noms des notes :

```
PEN 0; (20,80)-(43,95)
```

définit la case 0 formée par le rectangle (20,80)–(43,95)

```
PEN 1; (44,80)-(67,95)
```

définit la case 1 formée par le rectangle (44,80)–(67,95) et ainsi de suite...
Le tracé des cases et leur numérotation par l'instruction PEN peuvent se faire dans une boucle :

```
50 FOR N=0 TO 6
60   COL=20+N*24
70   BOX (COL,80)-(COL+24,95)
80   PEN N; (COL,80)-(COL+23,95)
90 NEXT N
```

D'une note à l'autre, N augmente de 1 et COL (numéro de la colonne de gauche de la case correspondante) augmente de 24 (trois caractères).

• Le branchement en une ligne différente pour l'appui sur chaque case est effectué par ON PEN :

```
150 ON PEN GOTO 200,210,220,230,240,250,260
```

• Et le jeu des notes, avec renvoi sur l'instruction ON PEN :

```
200 PLAY "DO" : GOTO 150
210 PLAY "RE" : GOTO 150
220 PLAY "MI" : GOTO 150
230 PLAY "FA" : GOTO 150
240 PLAY "SO" : GOTO 150
250 PLAY "LA" : GOTO 150
260 PLAY "SI" : GOTO 150
```

• Si vous exécutez le programme, vous constatez que chaque case donne bien la bonne note mais qu'en outre le reste de l'écran produit "DO". En effet l'instruction ON PEN renvoie à la ligne suivante lorsqu'on appuie en dehors des cases définies dans l'instruction PEN.
Rajoutons donc la ligne suivante:

```
160 GOTO 150
```

qui renvoie sur ON PEN lorsqu'on appuie en dehors des cases.

Cette fois vous pouvez jouer quelques notes...

Pianoter au clavier ON KEY= ... GOTO...

Les touches au clavier ne ressemblent guère à celle d'un piano mais avec un peu d'habitude et grâce à l'instruction ON KEY, nous allons pouvoir jouer directement dix notes différentes au clavier (*KEY*: touche).

La ligne suivante:

```
ON KEY="1" GOTO 200
```

signifie qu'à tout moment dans l'exécution du programme, l'appui sur la touche 1, provoquera le branchement à la ligne 200. Il suffit dans cette ligne 200 de faire jouer la note qu'on veut faire correspondre au chiffre 1 pour que chaque fois que l'utilisateur appuie sur 1, la machine joue cette note. Choisissons par exemple de faire correspondre les chiffres de 1 à 0 aux sept notes de l'octave 5 et trois dernières notes de l'octave 4:

1 correspond à O4 SO
2 correspond à O4 LA
3 correspond à O4 SI
4 correspond à O5 DO
5 correspond à O5 RE

6 correspond à O5 MI
7 correspond à O5 FA
8 correspond à O5 SO
9 correspond à O5 LA
0 correspond à O5 SI

Construisons le programme:

• déclarer que l'appui sur la touche 1 branche en ligne 210

2	220
3	230
	....
0	300

- attendre que l'utilisateur appuie sur une des dix touches

- jeu des notes

210 jouer la note et aller attendre
200 ....
....
300 ....

L'instruction d'attente sera simplement une boucle sans fin:

GOTO 200.

D'où le programme où vous apprécierez la possibilité de dupliquer une ligne en changeant seulement son numéro et quelques caractères:

```
5 ' PIANO CLAVIER
10 ONKEY="1" GOTO 210
20 ONKEY="2" GOTO 220
30 ONKEY="3" GOTO 230
40 ONKEY="4" GOTO 240
50 ONKEY="5" GOTO 250
60 ONKEY="6" GOTO 260
70 ONKEY="7" GOTO 270
80 ONKEY="8" GOTO 280
90 ONKEY="9" GOTO 290
100 ONKEY="0" GOTO 300
200 GOTO 200
210 PLAY "O4 SO" : GOTO 200
220 PLAY "O4 LA" : GOTO 200
230 PLAY "O4 SI" : GOTO 200
240 PLAY "O5 DO" : GOTO 200
250 PLAY "O5 RE" : GOTO 200
260 PLAY "O5 MI" : GOTO 200
270 PLAY "O5 FA" : GOTO 200
280 PLAY "O5 SO" : GOTO 200
290 PLAY "O5 LA" : GOTO 200
300 PLAY "O5 SI" : GOTO 200
```

Instructions vues

PLAY
PEN
ON PEN GOTO
ON KEY=GOTO

Chapitre 10
Fonctions et sous-programmes

Les fonctions

Il arrive que dans certains programmes, on soit amené à répéter souvent le même calcul simple. Dans ce cas on a intérêt à remplacer ce calcul par une fonction qui le définira une fois pour toutes.

Supposons que vous soyez géomètre. Vous avez à calculer beaucoup de distances. La fonction qui calcule la distance entre deux points de coordonnées (X1,Y1) et (X2,Y2) pourra s'écrire:

```
DEF FNDIST(X1,Y1,X2,Y2)=SQR( (X1-X2)^2 + (Y1-Y2)^2)
```

La définition d'une fonction commence toujours par DEF FN, la suite des caractères précise le nom que l'on veut donner à la fonction. Puis il faut indiquer entre parenthèses les arguments de la fonction et enfin, après le signe =, la manière de calculer sa valeur.

Calculons la distance par rapport à un point de coordonnées (100,100):

```
10 'DISTANCES
20 DEF FNDIST(X1,Y1,X2,Y2)=SQR ((X1-X2)^2+(Y1 -Y2)^2)
30 XA=100:YA=100
40 DO
50   INPUT "COORDONNEES DU POINT";XM,YM
60   PRINT FNDIST(XM,YM,XA,YA)
70 LOOP
```

L'appel de la fonction FNDIST (ligne 60) se fait comme pour n'importe quelle autre fonction. Evidemment, il faut mettre les valeurs des arguments dans le même ordre que lors de la définition. Utilisons cette fonction pour connaître tous les points qui sont à égale distance d'un autre.
Voici les lignes à modifier:

```
40 CLS: PSET (XA,YA)
50 FOR C=80 TO 120
60 FOR L=80 TO 120
70 IF FNDIST(XA,YA,C,L)<18 THEN PSET(C,L)
80 NEXT L
90 NEXT C
```

La méthode n'est pas excessivement rapide mais le résultat est intéressant ...

Les fonctions à définir ne sont pas limitées aux calculs numériques: on peut en fabriquer aussi qui manipulent des chaînes de caractères.
Par exemple, on a souvent besoin de connaître le premier caractère d'un mot ou d'une réponse. Dans le cas d'une réponse, cela permet à l'utilisateur de réduire sa frappe à une seule lettre.
On définira le premier caractère d'une chaîne comme suit:

```
DEF FNPREM$(C$)=LEFT$(C$,1)
```

Le nom de la fonction suit les mêmes règles que le nom d'une variable: il se termine par un $ si son résultat est une chaîne de caractères.

```
?FNPREM$("OUI")
O
```

La définition des fonctions se place généralement en début de programme; les fonctions ainsi définies sont alors utilisables n'importe où dans la suite du programme.

Les sous-programmes

Un sous-programme est une partie de programme autonome, qui effectue une ou plusieurs actions bien précises.

Vous êtes, par exemple, architecte et vous concevez des maisons individuelles. Pour tirer une vue de face de la maison, il faut représenter: le mur de façade, le toit, la (ou les) porte(s), les fenêtres.

Fixons la taille du mur de façade:

```
10 'MAISON
20 CLS
30 BOX(60,140)-(260,80)
```

Il faut maintenant représenter le toit: plusieurs toits sont possibles. Nous le mettons dans une partie de programme à part pour mieux l'identifier:

```
200 'TOIT 4 PENTES
210 LINE (50,80)-(270,80)
220 LINE         -(240,40)
230 LINE         -(80,40)
240 LINE         -(50,80)
```

Cette partie va devenir un sous-programme.
L'exécution du sous-programme est demandée par l'instruction GOSUB (*GO to SUBroutine*: aller au sous-programme):

```
110 GOSUB 200
```

Cette ligne signifie que la suite de l'exécution est en ligne 200. On termine le sous-programme par l'instruction RETURN:

```
250 RETURN
```

Ceci signifie: revenir à l'endroit où l'on a été appelé, ou plus exactement, à l'instruction qui suit l'instruction GOSUB qui a appelé le sous-programme.

Essayons un peu l'ébauche de la maison:

```
RUN
```

Le dessin se fait bien mais on obtient cependant un message d'erreur:

```
Return without Gosub in 250
```

L'instruction RETURN renvoie à l'instruction du programme qui suit immédiatement GOSUB. Dans le cas présent, le sous-programme a été exécuté une première fois, puis une deuxième fois sans avoir été appelé par GOSUB. Le deuxième passage sur RETURN provoque le déclenchement d'une erreur.
Il est absolument indispensable de séparer le sous-programme de la partie principale et d'indiquer par une instruction la fin du programme, c'est-à-dire de l'exécution:

```
190 END
```

Essayons à nouveau:

```
RUN
```

Ça marche.
Pour compléter la construction, il faut une porte. Si (C,L) désignent les coordonnées du milieu de la porte, en bas, on la trace avec le sous-programme suivant:

```
300 'PORTE
310 BOX(C,L)-(C-16,L-40)
320 BOX(C,L)-(C+16,L-40)
330 RETURN
```

Comme pour le précédent sous-programme, nous avons mis une ligne de commentaire en tête. Ceci n'est pas du tout obligatoire; un sous-programme peut commencer sur une ligne quelconque. Mais c'est tellement plus lisible quand le programme est fini qu'on ne saurait s'en passer!
Avant d'appeler le tracé de la porte, il faut fixer le point en bas au milieu:

```
120 C=160: L=140
130 GOSUB 300
```

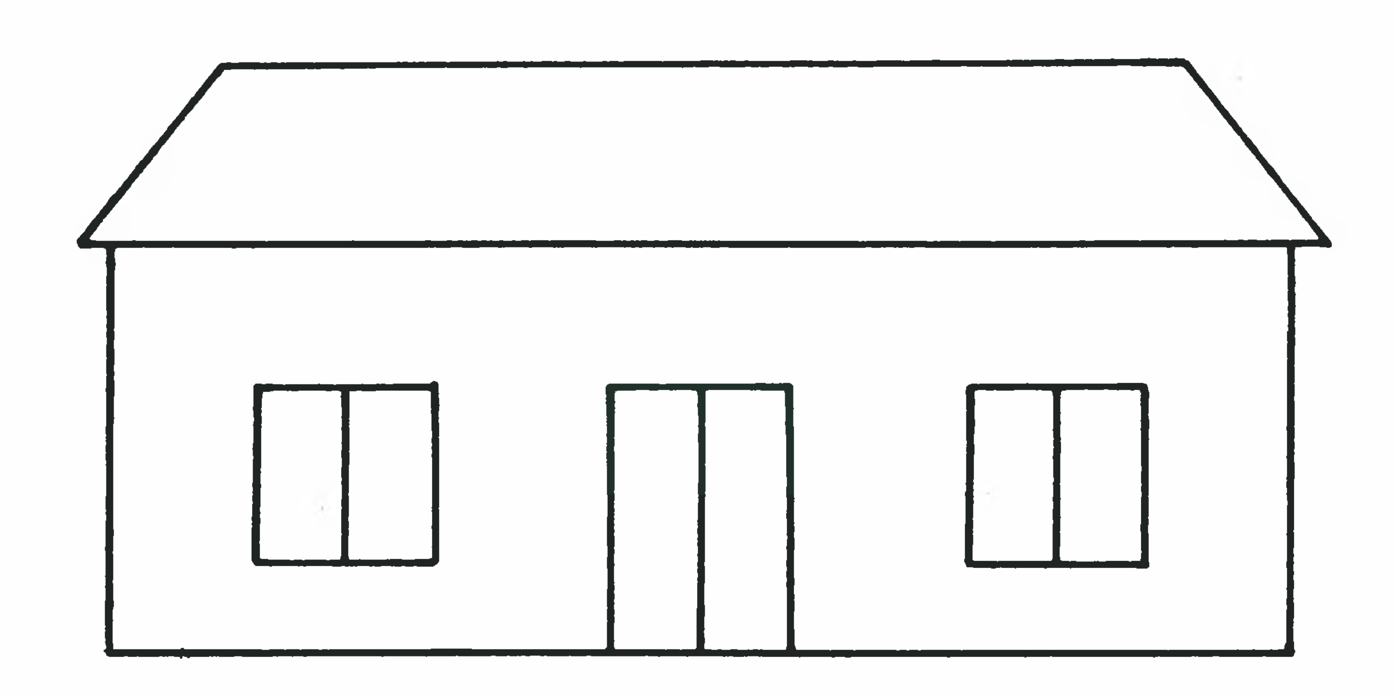
Reste à placer les fenêtres:

```
400 'FENETRE
410 BOX (C,L)-(C-16,L-30)
420 BOX (C,L)-(C+16,L-30)
430 RETURN
```

Pour mettre deux fenêtres, il suffit d'appeler deux fois de suite le sous-programme "fenêtre", en mettant les valeurs adéquates dans C et L:

```
140 C=100: L=130
150 GOSUB 400
160 C=220: L=130
170 GOSUB 400
```

La maison est finie. Vous pourriez l'agrémenter avec plusieurs types de portes, de fenêtres, de toits ou de façades. Mais attention, il faut tout de même respecter certaines contraintes quant aux dimensions relatives de chacune des parties.



Revenons sur le fonctionnement des sous-programmes.
A la rencontre d'une instruction GOSUB , l'interpréteur va effectivement à la ligne indiquée, comme avec l'instruction GOTO. Mais il se souvient de l'endroit d'où il est parti et lorsqu'il rencontre l'instruction RETURN, à la fin du sous-programme, il revient à l'instruction qui suit le GOSUB qui l'a appelé.
Dans le programme que nous venons d'étudier, voici les numéros des lignes qui ont été exécutées:

10 ┐
20 │ début du programme principal
100 │
110 ┘

200
210
220 sous-programme qui trace le toit
230
240
250

120 retour au programme principal
130

300
310 sous-programme qui trace la porte
320
330

140 retour au programme principal
150

400
.... et ainsi de suite

Les sous-programmes sont très utiles pour découper un programme en parties fonctionnelles: à chaque sous-programme, une fonction différente. On voit mieux où sont les instructions qui tracent le toit, celles qui tracent les fenêtres.... Et si l'on veut modifier la porte, on sait qu'il n'y a que le sous-programme "porte" à changer.
Les sous-programmes évitent aussi de recopier des parties trop fréquentes, ici les fenêtres. Qu'en serait-il pour une tour de trente-six étages...?

La trace d'un programme: TRON, TROFF

Si vous n'êtes pas convaincus par cette description du fonctionnement du programme, nous allons le vérifier immédiatement.
Il existe une instruction qui permet de connaître les numéros des lignes au fur et à mesure de leur exécution: TRON (*TRace ON*)
Tapez:

```
TRON
```

puis:

```
RUN
```

Vous voyez les numéros successifs s'afficher à l'écran, chacun dans une paire de crochets. Vous pouvez arrêter l'exécution par la touche STOP et reprendre par une touche quelconque. C'est très pratique pour comprendre le fonctionnement d'un programme... surtout quand il ne marche pas très bien.
Vous pouvez arrêter le flot des numéros de ligne par l'instruction opposée:

TROFF (*TRace OFF*).

Vous pouvez aussi le canaliser, c'est-à-dire ne tracer que les numéros des lignes qui vous intéressent:

```
TRON 400-430
```

ne visualisera que l'exécution du sous-programme "fenêtre".
Si vous trouvez que les numéros défilent un peu trop vite, vous pouvez en demander l'impression sur l'imprimante, vous aurez tout votre temps pour les examiner après:

```
TRON "LPRT:"
```

Des sous-programmes au choix: ON...GOSUB

Si vous étiez fabricant de fenêtres, il serait intéressant de pouvoir présenter vos différents modèles. Supposons que vous en ayez trois et que chacune soit tracée par un sous-programme. Avec l'instruction ON...GOSUB il suffit de donner le numéro de la fenêtre pour aller exécuter le sous-programme correspondant. D'abord les trois sous-programmes (simples):

```
200 'FENETRE1
210 BOX(C,L)-(C-16,L-30)
220 BOX(C,L)-(C+16,L-30)
230 RETURN

300 'FENETRE2
310 BOX(C-16,L-30)-(C+16,L)
320 LINE(C,L-30)-(C,L)
330 LINE(C-16,L-15)-(C+16,L-15)
340 RETURN
```

```
400 'FENETRE3
410 BOX(C-16,L-30)-(C+16,L)
420 LINE(C,L-30)-(C,L)
430 LINE(C-16,L-10)-(C+16,L-10)
440 LINE(C-16,L-20)-(C+16,L-20)
450 RETURN
```

Le programme principal doit, dans l'ordre:
— effacer l'écran
— demander le numéro du modèle
— tracer le modèle
Ecrivons-le:

```
10 'CHOIX DU MODELE
20 CLS
30 C=100: L=100
40 INPUT "QUEL MODELE (1,2 OU 3)";NB
50 ON NB GOSUB 200,300,400
60 END
```

En essayant le programme, vous constatez que si vous répondez 1, 2 ou 3 à la question, le programme dessine la fenêtre correspondante. Par contre, si vous répondez par un autre nombre, il ne dessine rien.
La traduction en français de ON NB GOSUB 200,300,400 est la suivante:
"si NB est égal à 1, alors aller au sous-programme 200
si NB est égal à 2, alors aller au sous-programme 300
si NB est égal à 3, alors aller au sous-programme 400
si NB est égal à une autre valeur, passer à la suite."

Cette instruction permet de rassembler en une seule ligne ce qui aurait été plus long à écrire avec IF...THEN
De manière générale, cette instruction peut être employée avec un nombre quelconque de sous-programmes.

Des appels indépendants: ON INTERVAL, ON KEY

Au cours d'un programme, on aimerait parfois faire deux choses à la fois. En toute rigueur, cela n'est pas possible mais il existe un moyen de s'en approcher.

Reprenons l'exemple du chronomètre. Il serait bien pratique d'afficher le chronomètre dans un jeu, pendant que les joueurs sont en action.
Cette possibilité est donnée par l'instruction ON INTERVAL... GOSUB. Nous allons l'introduire dans notre dernier programme, celui qui présente les modèles de fenêtres (si vous êtes un vendeur pressé il vous permettra de savoir combien de temps il vous faut pour présenter un modèle).
Le sous-programme de chronomètre, qui va être déclenché toutes les secondes, affichera ces secondes en haut à droite de l'écran:

```
1000 'CHRONO
1010 SEC=(SEC+1)MOD 60
1020 LOCATE 37,0:?SEC
1030 RETURN
```

Définissons maintenant le mécanisme de déclenchement:

```
15 ON INTERVAL=10 GOSUB 1000
```

Soit: tous les 10 intervalles de temps aller exécuter le sous-programme qui est en 1000. Or les intervalles de temps de l'horloge interne de l'ordinateur sont de 1/10e de seconde, donc le branchement au sous-programme se fait toutes les secondes et ceci quel que soit l'endroit du programme où l'on se trouve.
Il reste encore à mettre le chronomètre en route:

```
25 INTERVAL ON
```

Ceci déclenche le comptage des intervalles.
Avec cette instruction, vous pouvez, tout en utilisant un programme, faire sonner les heures, signaler que le temps est dépassé...

Le branchement sur un sous-programme peut se faire aussi au moyen des touches du clavier avec ONKEY...GOSUB . Le fonctionnement est le même que celui que nous avons utilisé pour le "piano" (chapitre 9).
Ces instructions peuvent servir à apporter de l'aide ou des renseignements à un utilisateur un peu perdu, par appui sur une touche (par exemple "?"):

```
100 ONKEY="?" GOSUB 5000
...
5000 'AIDE
5010?"VOICI QUELQUES RENSEIGNEMENTS....."
5020 RETURN
```

Instructions vues

DEF FN	END
GOSUB	ON INTERVAL...GOSUB
RETURN	ON KEY...GOSUB

Chapitre 11
Les tortues

Les “tortues” sont des figures qui, à l’instar de la tortue du langage Logo, peuvent apparaître en tout point de l’écran, avancer, reculer, tourner, dessiner, changer de taille, disparaître...
Vous pourrez concevoir leurs formes et en faire des objets animés, doués d’une grande vitesse et ne ressemblant plus guère à des tortues...

Faire apparaître et avancer une tortue: TURTLE, SHOW, FWD

Au départ une tortue est simplement représentée par un triangle isocèle. Appelons la première tortue en mode direct avec l’instruction TURTLE:

```
TURTLE 0
```

La première tortue est prête à se déplacer, à dessiner, ... mais invisible. Demandons-lui de se montrer:

```
SHOW 1
```

Le triangle apparaît au centre de l’écran, position de départ standard pour les tortues. Si vous changez SHOW 1 en SHOW 0, la tortue redevient invisible.
Une fois revenue, elle avancera avec l’instruction FWD (*forward*) suivie de la quantité dont elle doit avancer (ou reculer si le nombre est négatif):

```
FWD 50
```

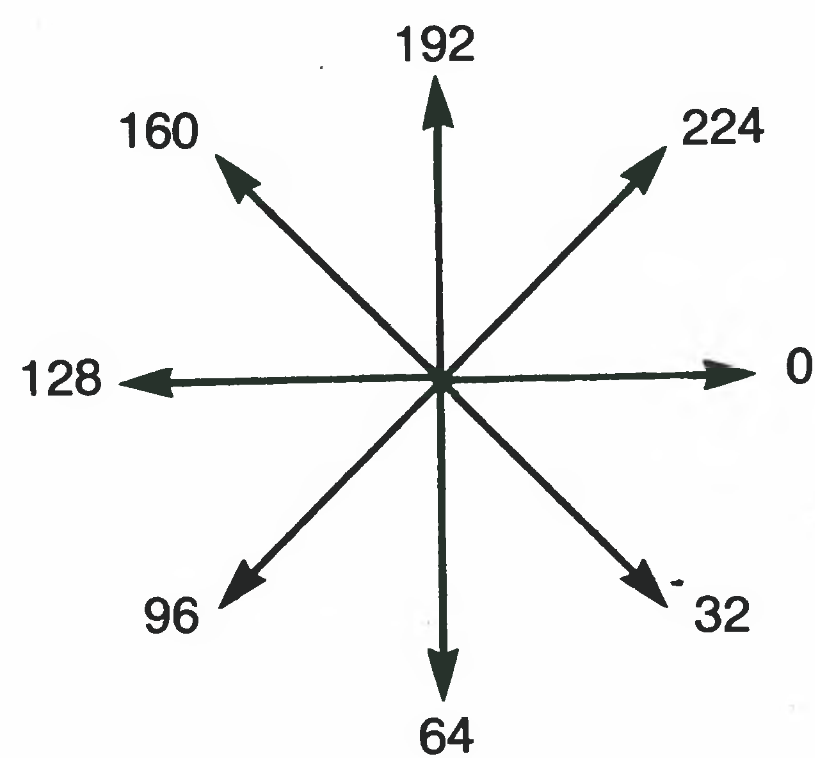
Le déplacement se fait vers la droite, direction du nez de la tortue, et est très rapide. En insérant un signe – devant le nombre 50, et en validant la nouvelle instruction, la tortue revient à son point de départ.

Dessiner avec la tortue: TRACE, HEAD

En se déplaçant la tortue peut baisser son crayon: l'ordre correspondant est TRACE 1, et lever son crayon avec TRACE 0. Voici le programme qui appelle la première tortue, la montre, lui fait baisser le crayon et enfin la fait avancer de 50 points:

```
10 'TORTUE
20 CLS
30 TURTLE 0
40 SHOW 1
50 TRACE 1
60 FWD 50
RUN
```

A peine apparue au centre de l'écran, la tortue a déjà avancé en laissant sa trace. Remarquez que le crayon est situé au centre de la tortue. En remplaçant TRACE 1 par TRACE 0, la tortue avance sans dessiner.
La direction du mouvement est modifiable par l'instruction HEAD (tête) suivie d'un nombre précisant de combien il faut tourner: un tour complet (360°) correspond à 256, un demi-tour (180°) à 128, un quart de tour (90°) à 64, un huitième de tour (45°) à 32 et ainsi de suite.



Ainsi HEAD 64 signifie: prendre la direction à 90° vers la droite et HEAD −64 : prendre la direction à 90° vers la gauche.
Ajoutons les deux lignes suivantes au programme:

```
70 HEAD 64
80 FWD 80
```

Essayez HEAD 32, HEAD −64... en ligne 70.

Faisons dessiner un rectangle par la tortue:

```
10 ' RECTANGLE
20 CLS
30 TURTLE 0
40 SHOW 1
50 TRACE 1
60 FWD 100: HEAD 64
70 FWD 50 : HEAD 64
80 FWD 100: HEAD 64
90 FWD 50 : HEAD 64
```

Rapide pour une tortue ...
A la fin du programme, la tortue est revenue dans sa position initiale. Cela est en général préférable.

Un octogone et des variantes

Lorsque la tortue sert à dessiner, il est inutile qu'elle se montre. Dessinons par exemple un octogone: il sera formé de segments de même longueur, de directions successives décalées de 45°.

```
10 'OCTOGONE
20 CLS
30 TURTLE 0
40 TRACE 1
50 FOR N=1 TO 8
60   FWD 30: HEAD 32
70 NEXT N
```

En remplaçant HEAD 32 par HEAD 16 et la valeur maximale de N par 16, on obtient un polygone de seize côtés . Mais une partie de la figure est en dehors de l'écran. Il est possible de déplacer le point de

départ de la tortue en précisant les coordonnées du point de départ désiré dans l'instruction TURTLE après le numéro de la tortue:

```
30 TURTLE 0,160,20
```

Vous pouvez essayer d'autres valeurs dans l'instruction HEAD: 48, 96, 112...

Faire tourner la tortue sur elle-même: ROT

Lorsque la tortue se déplace, elle garde la tête obstinément dirigée vers la droite, quelle que soit sa direction: l'instruction HEAD change sa direction mais pas son orientation propre. C'est l'instruction ROT qui la fait tourner autour de son centre. Cette instruction est suivie d'un nombre qui précise de combien doit tourner la tortue par rapport à son orientation précédente.
Ainsi pour que la tortue suive l'orientation de son trajet, il suffit de faire suivre chaque instruction HEAD par une instruction ROT de même valeur.
Montrons la tortue dessinant l'une des figures précédentes, par exemple l'octogone:

```
45 SHOW 1
60   FWD 30: HEAD 32: ROT 32
```

Et pour la faire disparaître à la fin:

```
80 SHOW 0
```

Avec de petits segments et de petits angles, la tortue semble décrire un cercle. Tournons la tortue le nez vers le centre de sa trajectoire et nous aurons un modèle de la Lune tournant autour de la Terre: elle met le même temps pour faire un tour sur elle-même que pour faire un tour autour de la Terre et en conséquence on en voit toujours la même face ... comme la tortue du programme suivant:

```
10 ' LUNE
20 CLS
30 TURTLE 0,160,20
40 SHOW 1
50 ROT 64
60 DO
70   FWD 4: HEAD 3: ROT 3
80 LOOP
```

Essayons maintenant de faire tourner la tortue sur elle-même:

```
10 'TOURNE
20 CLS
30 TURTLE 0,30,100
40 SHOW 1
50 DO
60   ROT 10
70 LOOP
RUN
```

Rien ne se passe. En effet l'instruction ROT n'a d'effet que lorsque la tortue se déplace, c'est-à-dire à la première instruction FWD rencontrée. Pour que l'instruction ROT agisse immédiatement, il faut modifier l'instruction SHOW 1 en SHOW 1,1. Le deuxième chiffre 1 signifie: prendre en compte immédiatement toutes les modifications (d'orientation, de taille...) demandées à la tortue. S'il n'est pas là, ces modifications ne sont prises en compte qu'à la première instruction FWD rencontrée.
Modifions donc la ligne 40 du programme:

```
40 SHOW 1,1
```

En même temps qu'elle tourne, la tortue peut avancer:

```
65 FWD 3
```

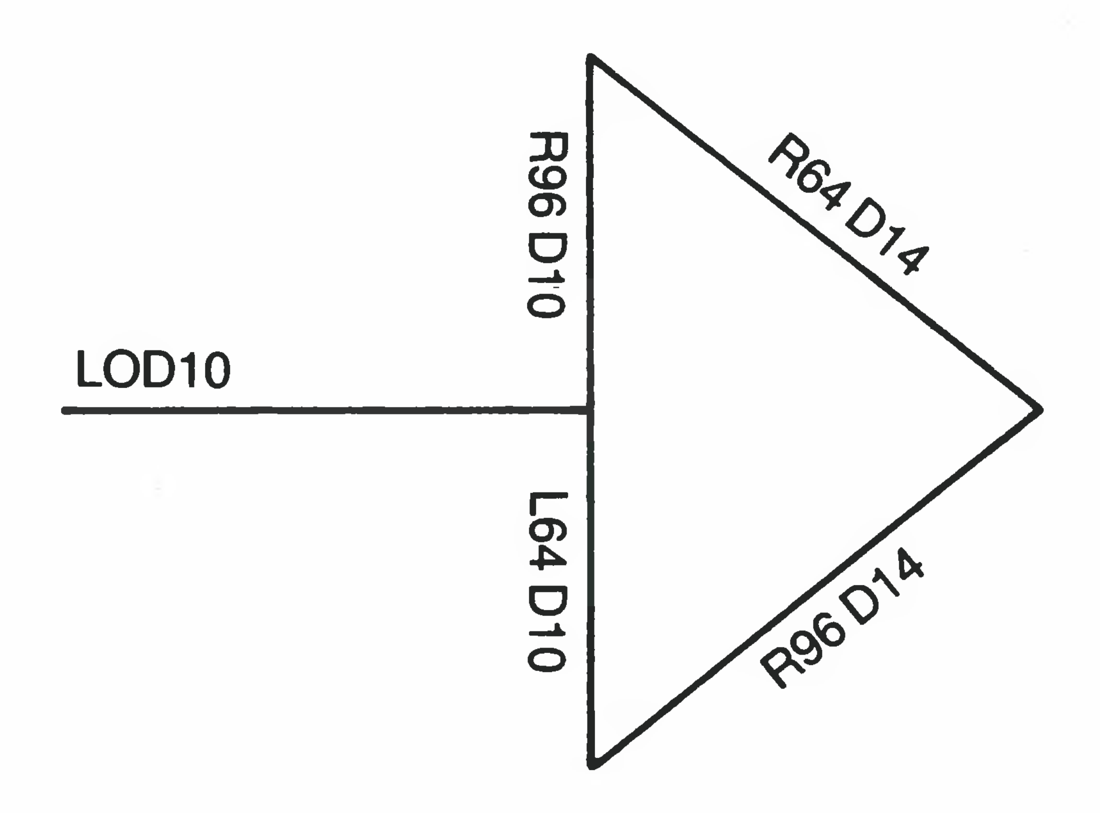
et ceci dans une direction quelconque:

```
45 HEAD 32
```

Il faut bien distinguer:
— le déplacement de la tortue réglé par FWD et HEAD,
— son orientation propre réglée par ROT

Une nouvelle tortue

Créons une deuxième tortue de forme différente:



Elle sera définie dans l'instruction TURTLE par une chaîne de caractères précisant la suite des segments nécessaires pour dessiner cette tortue. Pour la distinguer de la première, nous allons lui donner un numéro différent: 2. Ce numéro est indiqué immédiatement après TURTLE:

```
10 ' TORTUE2
20 CLS
30 TURTLE 2,,,"L0D10L64D10R96D14R64D14R96D10"
40 SHOW 1,1
```

La chaîne définissant une tortue est formée d'une suite de doublets qui correspondent aux segments. Chaque doublet comprend deux éléments: le premier indique de combien il faut tourner, le second de combien il faut avancer. Les deux éléments sont obligatoires.

L0D10 signifie: — tourne à gauche (*Left*) de 0
— avance en traçant (*Down*) de 10

R96D14 signifie: — tourne à droite (*Right*) de 96
— avance en traçant (*D*) de 14

Faisons tourner cette nouvelle tortue:

```
50 DO
60   ROT 10
70 LOOP
```

La rotation se fait autour du point de départ A du tracé. Avec le triangle de la tortue standard, la rotation se faisait autour du centre du triangle.
Pour faire tourner notre tortue sur elle-même autour d'un autre point O plus central, rajoutons le doublet L128U15 qui signifie:
— change l'orientation de départ de 0 en 128
— avance sans tracer (*Up*) de 15

```
30 TURTLE2,,,"L128U15L128D10L64D10R96R64D14R96D10"
```

Cette fois la tortue tourne autour du point O. Ce point de départ du tracé d'une tortue est aussi la position de son "crayon".
En résumé quatre types de doublets sont possibles dans la chaîne de définition d'une tortue: LxDy, LxUy, RxDy, RxUy.

La guerre des tortues

Reprenons la tortue précédente, dans sa première version, le point de départ étant le point A. Plaçons-la à gauche de l'écran et adjoignons lui une comparse symétrique placée à droite de l'écran:

```
10 ' TORTUE2
20 CLS
30 TURTLE2,20,100,"L0D10L64D10R96D14R64D14R96D10"
40 SHOW 1,1
50 TURTLE 3,300,100,"L0D10L64D10R96D14R64D14R96D10"
60 SHOW 1,1
70 HEAD TO 128: ROT TO 128
```

Les instructions HEAD et ROT quand elles sont suivies de TO fixent la direction et l'orientation de la tortue à une valeur absolue. Ici la valeur 128 signifie simplement que la deuxième tortue opère un demi-tour par rapport à la position de départ.
Les instructions SHOW, TRACE, FWD... ne sont relatives qu'à une seule tortue à la fois, la dernière appelée par une instruction TURTLE.
Faisons avancer les deux tortues l'une vers l'autre:

```
100 DO
110   TURTLE 2
120   FWD 5
140   TURTLE 3
150   FWD 5
180 LOOP
```

Les deux tortues se croisent sans dommages. Pour leur donner un peu plus de matérialité, arrêtons-les lorsqu'elles se rencontrent.
La fonction INPUTTURTLE permet de savoir la position de la tortue, ou plus précisément les coordonnées du point de départ de son tracé.
Arrêtons donc les tortues lorsque leurs extrémités sont séparées par une distance inférieure au double de leur longueur:

```
130 INPUTTURTLE C2,L2
160 INPUTTURTLE C3,L3
170 IF C3 – C2<45 THEN EXIT
```

Un effet de ZOOM

Grâce à l'instruction ZOOM, chaque tortue peut devenir géante ou minuscule.
La taille de chaque tortue peut varier continûment sur une échelle de 0 à 255:

16 correspond à l'échelle 1 c'est-à-dire la taille standard
32 correspond à l'échelle 2
48 correspond à l'échelle 3
....
255 correspond à l'échelle 16

Au départ la tortue a donc la taille définie par le nombre 16; ceci ne signifie pas que sa longueur soit 16, l'échelle étant conventionnelle.
Essayons l'effet de l'instruction ZOOM qui, comme l'instruction ROT, ne prend effet immédiatement que si elle est précédée par SHOW 1,1:

```
10 ' ZOOM
20 CLS
30 TURTLE 1
40 SHOW 1,1
50 TURTLE 2
60 SHOW 1,1
70 ZOOM 32
```

La deuxième tortue, qui a subi l'instruction ZOOM 32, est trois fois plus grande que la tortue standard. En effet le nombre qui suit ZOOM s'ajoute à la taille précédente de la tortue (repérée sur l'échelle 0-256) pour donner sa nouvelle taille. Ici la taille originale est 16, il s'y ajoute 32 et la taille finale est 48, ce qui correspond à l'échelle 3.

Le grossissement peut être continu :

```
50 DO
60   ZOOM 4
70 LOOP
```

Pour arrêter le grossissement avant que la tortue ne dépasse de l'écran, utilisons la fonction ZOOM qui a le même nom que l'instruction (mais ne doit pas être confondue avec elle) et donne la taille de la tortue. Si la tortue est à sa taille maximale :

```
? ZOOM
255
```

Rajoutons le test suivant pour interrompre le grossissement continu :

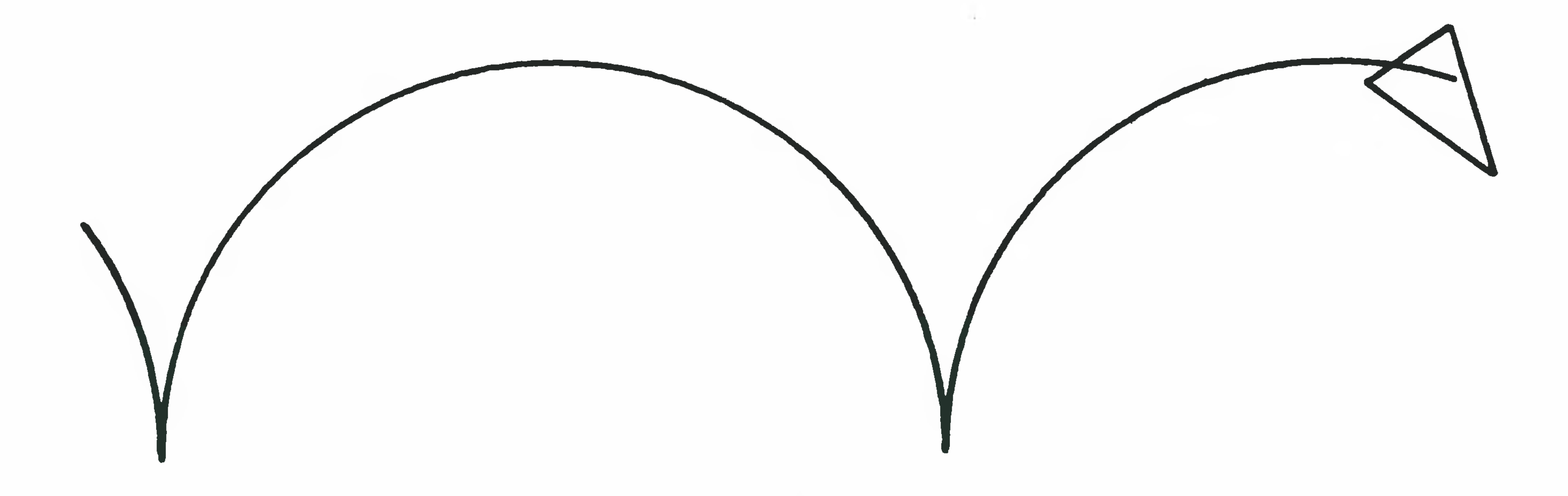
```
65 IF ZOOM > 80 THEN EXIT
```

Le zoom peut aussi être négatif : grossissons d'un coup la tortue avant de la diminuer continûment.

```
50 ZOOM 80
60 DO
65   ZOOM -4
70 LOOP
```

Le rebond : fonction HEAD

Pour finir faisons une petite simulation de mouvement avec la tortue, par exemple une série de rebonds en demi-cercles.



La tortue part du bord gauche de l'écran et doit d'abord tracer un cercle:

```
10 ' REBOND
20 CLS
30 TURTLE 1,0,100
40 SHOW 1,1
50 TRACE 1
60 DO
70   FWD 5: HEAD 4: ROT 4
90 LOOP
```

Pour arrêter la tortue lorsqu'elle est verticale, utilisons la fonction HEAD (même nom que l'instruction) qui donne la valeur de la direction de la tortue, de même que la fonction ZOOM donne sa taille:

```
80 IF HEAD = 68 THEN HEAD 124
```

Cette ligne signifie: lorsque la direction de la tortue atteint 68 alors elle doit tourner de 124.
Lorsque, en fin de ligne 70, la tortue vient de tracer un segment vertical, sa direction est déjà 68 puisque l'augmentation par l'instruction HEAD se fait après le tracé par FWD. Ainsi le dernier segment tracé correspond à HEAD=68. Il suffit alors d'ajouter 124 à la direction de la tortue pour qu'elle se retrouve verticale dans l'autre sens.

Avec ce principe, vous n'aurez plus de difficultés pour transformer la tortue en balle de squash!

Fonctions et Instructions vues

TURTLE
SHOW
HEAD
ROT
FWD
TRACE
INPUTTURTLE

Chapitre 12
Données et tableaux

Des constantes dans un programme

Si vous voulez étudier la répartition des âges, des tailles d'un groupe d'individus, des prix d'un produit dans une région..., vous aurez une série de nombres à entrer dans la mémoire de l'ordinateur.
Lorsqu'on doit utiliser un grand nombre de données, les instructions DATA, READ simplifient l'écriture.

Les nombres, par exemple le prix d'un produit en divers points de vente, peuvent être inscrits les uns à la suite des autres dans le programme avec l'instruction DATA (données):
100 DATA 52,48,50,51,49,53,47

Pour les utiliser, il suffit de les lire successivement avec l'instruction READ. Voici le programme qui se contente de lire et d'afficher la suite des données:

```
10 ' PRIX
20 FOR I=1 TO 7
30   READ PRIX
40   PRINT "PRIX";I;":";PRIX
50 NEXT I
100 DATA 52,48,50,51,49,53,47
```

La ligne 30 lit la première donnée 52 dans l'instruction DATA et la place dans la variable PRIX.
La ligne 40 affiche la valeur de la variable PRIX.
Au deuxième tour, la ligne 30 lit la deuxième donnée dans la liste DATA , la ligne 40 l'affiche et ainsi de suite jusqu'à la septième donnée qui est lue et affichée au septième tour.

Il est maintenant possible de travailler avec ces données par exemple de calculer la valeur moyenne des prix relevés.
La somme des prix, SOMME, sera calculée à chaque tour: au premier tour, la somme est simplement la première donnée, au deuxième tour on ajoute la deuxième donnée et ainsi de suite jusqu'à la dernière donnée:

```
45 SOMME = SOMME + PRIX
```

Le calcul de la moyenne se fait lorsque la boucle est terminée:

```
60 PRINT "MOYENNE="; SOMME/7
```

Une condition indispensable est à respecter: que le nombre des données dans l'instruction DATA soit au moins égal au nombre de lectures prévues par l'instruction READ.
Si vous supprimez la dernière donnée: 47, l'exécution provoque le message:

```
Out Of Data in 30
```

Il en manque ...

Pour ne pas demander la lecture d'un nombre de données supérieur à celui existant, on peut, comme dans l'exemple précédent, compter ces données. Mais souvent le nombre de données varie, ainsi des prix peuvent être rajoutés à la liste au fur et à mesure des relevés. Pour éviter d'avoir à tenir à jour le compte de ces données, il suffit d'indiquer en fin de liste par un nombre complètement différent des données (par exemple 0 pour un prix) que la liste est terminée:

```
20 DO
30   READ PRIX
35   IF PRIX=0 THEN EXIT
40   SOMME = SOMME + PRIX
50 LOOP
```

Lorsque l'instruction READ rencontre la donnée 0, il y a sortie de la boucle.
Ceci permet une grande souplesse dans la gestion des données.

Des données chaînes de caractères

Une liste de données peut contenir non seulement des nombres, mais aussi des chaînes de caractères, voire un mélange de nombres et de chaînes.
Voyons le cas d'une suite de données chaînes de caractères, par exemple une liste de noms avec leurs numéros de téléphone. Bien sûr le nom des variables (nom ou numéro) doit se terminer par $: NOM$ et TEL$.

Les données seront alternativement: un nom, son numéro, un nom, son numéro .... Elles peuvent être réparties dans une ou plusieurs instructions DATA, sachant qu'une instruction DATA, comme toute ligne de programme, ne peut contenir plus de 255 caractères.
Pour faciliter la lecture, présentons les données sous la forme d'une instruction DATA pour un nom et son numéro:

```
DATA LUCIEN,111-11-11
```

Les guillemets pour les chaînes de caractères sont facultatifs dans l'instruction DATA.

Voici un programme qui affiche la liste des noms suivis de leurs numéros :

```
10 'TELEPHONE
20 CLS
30 DO
40    READ NOM$,TEL$
50    IF NOM$="ZZZ" THEN EXIT
60    PRINT NOM$,TEL$
70 LOOP
100 DATA CECILE , 111-11-11
110 DATA CLEMENCE , 222-22-22
120 DATA NICOLAS , 333-33-33
130 DATA LUCIEN , 444-44-44
140 DATA BAPTISTE , 555-55-55
999 DATA ZZZ,0
```

La lecture des données se fait deux par deux, et la fin de la liste est précisée par le nom improbable "ZZZ". Ainsi il est possible d'ajouter des noms à la liste, entre les lignes 140 et 999, sans modifier le programme.

Ce programme affiche à chaque exécution la liste de tous les noms, dans l'ordre où ils ont été rentrés. Transformons-le en un programme de recherche du numéro correspondant à un nom donné.
Il faut:
— demander à l'utilisateur le nom X$ dont il veut le numéro,
— lire la liste de données jusqu'à trouver ce nom,
— l'afficher avec son numéro,
— afficher "absent" si on arrive à "ZZZ" sans avoir trouvé le nom X$.

```
10 ' TELEPHONE
20 CLS
25 INPUT "QUEL NOM VOULEZ-VOUS"; X$
30 DO
40   READ NOM$,TEL$
50   IF NOM$=X$ THEN PRINT NOM$,TEL$ : EXIT
60   IF NOM$="ZZZ" THEN PRINT "ABSENT" : EXIT
70 LOOP
```

Les données (lignes 100-999) restent identiques. La boucle de lecture des données possède deux sorties possibles:
— en ligne 50 si le nom cherché a été trouvé,
— en ligne 60 si la liste a été lue jusqu'à "ZZZ" sans que le nom cherché ait été trouvé.
Pour permettre à l'utilisateur de demander autant de noms qu'il veut, répétons l'opération avec une deuxième boucle qui enserre la première:

```
22 DO
80 LOOP
RUN
```

Il y a un problème: après un ou deux noms, un message d'erreur "Out Of Data In 40" apparaît. En effet l'instruction READ lit les données dans l'ordre et aboutit rapidement à la fin de la liste. Pour que la lecture reprenne au début de la liste, il faut rajouter l'instruction RESTORE à la fin de chaque lecture de toute la liste:

```
75 RESTORE
```

Et enfin, permettons à l'utilisateur d'arrêter le programme en appuyant simplement sur ENTREE en réponse à la question "Quel nom voulez-vous".

```
25 INPUT "QUEL NOM VOULEZ-VOUS";X$
27 IF X$="" THEN EXIT
```

Les tableaux de données

Lorsqu'on veut pouvoir accéder directement à une donnée ou la modifier sans avoir à relire toute la liste, on donne des numéros aux données et la 4e, la 10e ou la 500e peuvent ainsi être trouvées directement. Ce faisant, on constitue un "tableau" de données.
Si nous voulons faire quelques statistiques sur la météo, la première chose à faire sera d'enregistrer régulièrement les températures.
Prenons la liste des températures d'un mois et rangeons-la dans un tableau, dont le nom sera TEMP, en donnant à chaque température un numéro: le numéro du jour dans le mois.
TEMP(12) représentera la température du 12e jour.
Le tableau comportera donc 31 "éléments" numérotés de 1 à 31.
Au préalable il faut réserver la place nécessaire en mémoire à ces éléments en "déclarant" la dimension du tableau, c'est-à-dire le nombre de ses éléments, avec l'instruction DIM:

```
10 'METEO
20 DIM TEMP(31)
```

La ligne 20 déclare un tableau de 31 éléments (en fait 32 car l'interpréteur Basic compte toujours à partir de 0).

Pour rentrer les données dans un tableau, c'est-à-dire affecter des valeurs à chaque élément, deux solutions sont possibles: soit par interrogation de l'utilisateur (INPUT), soit par lecture d'une liste (DATA , READ).

Dans le premier cas les données ne restent dans le tableau que pour la durée d'une exécution, à chaque exécution les éléments sont remis à zéro, dans le deuxième les éléments restent en permanence dans le programme.

Voyons d'abord l'interrogation pour préciser les valeurs des éléments du tableau TEMP:

```
30 CLS
40 FOR I=1 TO 31
50   PRINT "TEMPERATURE DU";I;
60   INPUT TEMP(I)
70 NEXT I
```

Le point-virgule en fin de ligne 50 implique l'affichage du point d'interrogation de l'instruction INPUT après le numéro I.
Lors de l'exécution vous devez répondre à chaque question par la donnée correspondante.

La deuxième solution est de lire les données dans une instruction DATA:

```
30 CLS
40 FOR I=1 TO 31
50   READ TEMP(I)
60 NEXT I
99 DATA 25,27,26,30,29,33,28,32,31,33,29,28,31,30,26,29,
   32,30,28,25,28,26,31,30,27,29,28,29,27,26,29
```

Quelle est l'utilité d'un tableau ainsi constitué? D'abord de pouvoir aller chercher la température du jour voulu (ce pourrait être le prix de l'article n° 45, le score du joueur n° 3 ....).
Mais un tableau peut aussi être utile pour des opérations plus complexes: faire la moyenne des éléments, chercher quel est le plus grand, le plus petit, les classer par ordre croissant, etc.

Ecrivons par exemple le programme qui détermine les températures maximale et minimale du mois étudié. La fonction MIN permet de trouver facilement le minimum: MIN(A,B) est le plus petit de A ou B.
MIN(3,5) vaut 3
De même MAX(A,B) est le plus grand de A ou B:
MAX(3,5) vaut 5

Soient TMIN et TMAX les deux températures cherchées; nous ne pouvons pas choisir un nom de variable qui commence par MIN ou MAX puisque ce sont des fonctions du Basic.
Prenons comme valeurs initiales pour ces variables des valeurs improbables: TMIN=100, TMAX=−50.

```
100 ' MINI-MAXI
110 TMIN=100: TMAX=-50
120 FOR I=1 TO 31
130   TMIN=MIN(TEMP(I), TMIN)
140   TMAX=MAX(TEMP(I), TMAX)
150 NEXT I
160? "MAXIMUM";TMAX
170? "MINIMUM";TMIN
```

Au premier tour de boucle la ligne 130 compare la température du 1er jour, TEMP(1), avec TMIN qui vaut 100 et affecte TEMP(1) à TMIN. De même TMAX devient TEMP(1).

Au 2e tour TEMP(1) et TEMP(2) sont comparées, la plus petite des deux affectée à TMIN et la plus grande à TMAX ... et ainsi de suite jusqu'à TEMP(31).

Nous pouvons aussi tracer le graphique qui représente l'évolution de la température dans le mois: horizontalement seront portés les jours de 1 à 31 et verticalement les températures correspondantes.
L'échelle horizontale doit permettre de placer 31 jours sur 320 colonnes; prenons 10 colonnes entre 2 jours.
L'échelle verticale doit permettre d'atteindre environ 35° avec 200 lignes; prenons 5 lignes pour 1°.

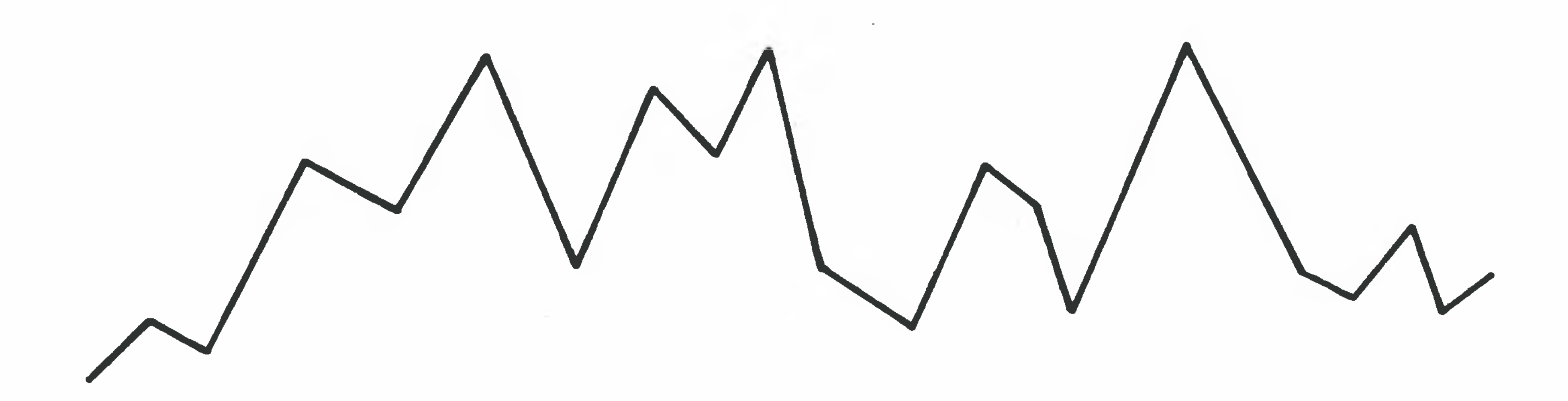
Le premier point aura pour coordonnées C=0 et L=200−TEMP(1)*5.

Et pour le point n° I:

```
C = (I−1)*10
L = 200−TEMP(I)*5
```

D'où le programme:

```
200 ' COURBE
210 PSET (0, 200−TEMP(1)*5 )
220 FOR I=2 TO 31
290   LINE −( (I−1)*10 , 200−TEMP(I)*5 )
```



Les tableaux à plusieurs dimensions

Si vous voulez faire une étude plus poussée de la météo de la région, vous pourrez utiliser les données de plusieurs stations météo. Pour noter ces nouvelles données, nous allons ajouter une dimension au tableau: chaque température sera repérée par deux indices, le premier représentera le numéro de la station météo et le deuxième le jour du mois comme précédemment.

Ainsi si nous étudions quatre stations,
TEMP(1,5) est la température relevée à la 1ere station le 5e jour
TEMP(2,5) est la température relevée à la 2e station le 5e jour
TEMP(4,30) est la température relevée à la 4e station le 30e jour

La taille du tableau, c'est-à-dire la valeur maximale de chaque indice, doit encore être déclarée au début du programme avec l'instruction DIM .

Quant à l'entrée des données, il faudra répéter l'opération précédente pour chaque station:

```
10 ' STATIONS
20 CLS
30 FOR STATION =1 TO 4
40   FOR JOUR=1 TO 31
50      READ TEMP(STATION,JOUR)
60   NEXT JOUR
70 NEXT STATION
99 DATA ......
```

Une fois les données entrées en machine, on peut se livrer à des comparaisons entre les stations: celle qui a le plus fort maximum, la plus grande moyenne.....

Tableaux de chaînes

Les chaînes de caractères peuvent, comme les nombres, être rangées dans des tableaux.
Le nom d'un tableau chaîne doit, comme pour les variables chaînes, se terminer par $ et sa taille doit être déclarée avant de le remplir.

Si vous avez quelques difficultés à retenir des dates, nous pouvons construire un tableau dont les éléments seront des événements importants et l'indice, l'année correspondante. Ainsi E$(1900)

représentera l'événement choisi pour l'année 1900.
Plaçons d'abord quelques événements dans le tableau après avoir réservé de la place pour 2000 éléments de 0 à l'an 2000:

```
10 ' DATES
20 DIM E$(2000)
30 ' Quelques dates
40 E$(1543)="Copernic: la Terre tourne"
50 E$(1747)="Franklin invente le paratonnerre"
60 E$(1826)="Niepce invente la photographie"
```

A vous de remplir le tableau...
Le programme va ensuite demander à l'utilisateur quelle année l'intéresse et lui donner l'événement correspondant. Si cette année n'a pas reçu d'événement, la réponse sera "Rien à signaler" :

```
200 ' INTERROGATION
210 DO
220    INPUT "QUELLE ANNEE"; AN
230    REP$=E$(AN)
240    IF REP$="" THEN REP$="Rien à signaler"
250    ?REP$
260 LOOP
```

Permettons à l'utilisateur de fournir à son tour un événement de son choix pour une année vide:

```
255 IF REP$="" THEN GOSUB 300
290 END
300 ' Remplissage
310 LINEINPUT "Quel événement important"; E$(AN)
320 RETURN
```

Si l'utilisateur n'a aucun événement à proposer, il lui suffit de donner une réponse vide en appuyant sur ENTREE.
Mais ces réponses introduites en cours de programme par une instruction INPUT ne sont utilisables que pour une exécution. Une nouvelle exécution efface ces données. Pour remédier à ceci, une solution, celle étudiée dans le chapitre suivant: la conservation des données dans un fichier.

Fonctions et instructions vues

READ
DATA
RESTORE
DIM
MIN
MAX

Chapitre 13
Les fichiers sequentiels

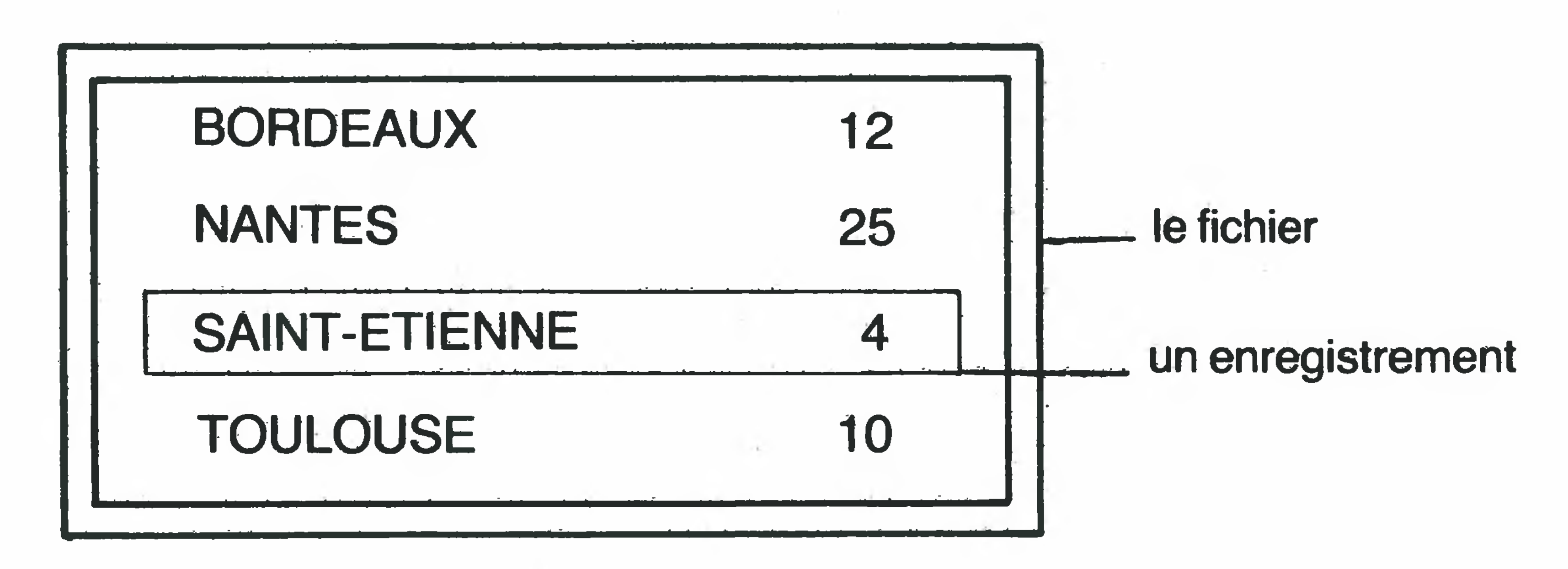
Qu'est-ce qu'un fichier de données?

Les programmes que vous avez enregistrés sur disquette jusqu'à présent constituent la première catégorie de fichiers, celle des "fichiers-programmes". Leur gestion est très facile.
Une deuxième catégorie de fichiers est celle des "fichiers de données".
Un fichier de données est constitué d'une liste de mots (chaîne de caractères) et/ou de nombres: liste de dépenses, de noms, d'adresses, de résultats de jeux...
Stocker ses données dans un fichier plutôt que dans un programme présente plusieurs avantages: d'abord on peut stocker beaucoup plus de données sur disquette, et ensuite un même fichier de données peut servir dans plusieurs programmes différents.
Notons d'abord qu'il est impossible d'enregistrer, de charger et encore moins d'exécuter un fichier de données avec une seule instruction comme on le fait avec un fichier-programme (SAVE, LOAD, RUN).

C'est un paquet inerte.
Pour gérer ce paquet de données, l'enregistrer, le relire à volonté, modifier une des données... vous pouvez utiliser deux procédés différents. Dans ce chapitre nous allons voir le premier de ces procédés: il consiste à enregistrer (et de relire) toutes les données les unes à la suite des autres sur la disquette (ou la cassette). C'est ce qu'on appelle constituer un "fichier à accès séquentiel".
Le deuxième procédé, que nous étudierons dans le prochain chapitre, permet d'enregistrer (et relire) directement n'importe quelle donnée du fichier. Le fichier ainsi constitué est alors un "fichier à accès direct".

La gestion d'un fichier séquentiel

Commençons donc par le type de fichiers le plus simple: les fichiers séquentiels. Si vous voulez conserver la liste de vos amis avec le total de leurs points au fur et à mesure de vos parties de bridge, de belote ou de tarot, comptabiliser le nombre de buts marqués au cours de la saison par toutes les équipes d'une division..., vous aurez une liste dont chaque élément sera constitué d'une chaîne de caractères et d'un nombre:



Chacun des éléments constitue un "enregistrement". Un enregistrement peut être plus compliqué. Par exemple le fichier des salariés d'une entreprise, qui sert, entre autres, à établir les bulletins de paie, contient, pour chaque employé: les nom, prénoms, adresse, numéro de sécurité sociale, salaire brut, retenues...
Pour utiliser votre fichier, il vous faudra en général deux programmes qui rempliront deux fonctions différentes. La première consiste à enregistrer le fichier sur la disquette, c'est-à-dire le transférer de la mémoire centrale sur la disquette: c'est l'écriture.

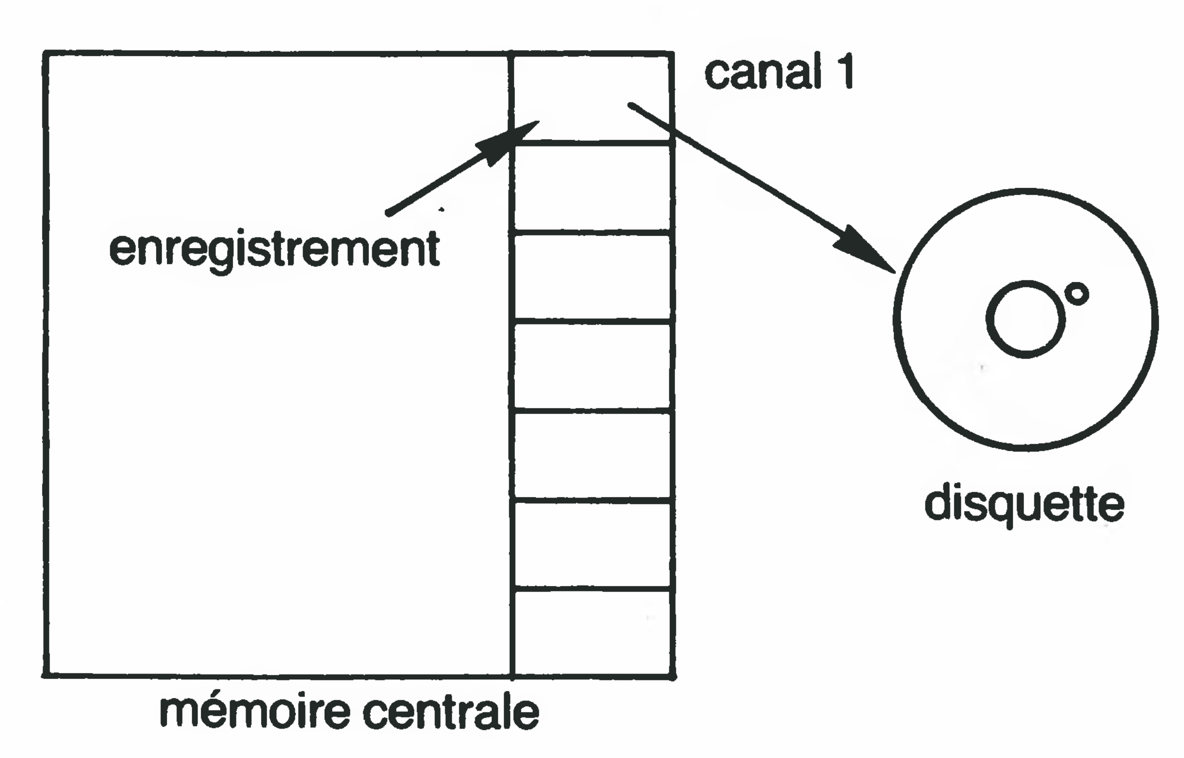
La deuxième opération, inverse, consiste à transférer le fichier de la disquette dans la mémoire centrale et à l'afficher à l'écran (ou sur imprimante): c'est la lecture.
Ces deux fonctions pourront d'ailleurs ultérieurement être réunies dans le même programme.

Attention: lire et écrire ont toujours pour sujet implicite l'"ordinateur" Il faut toujours le sous-entendre et ne pas rapporter ces opérations de lecture et d'écriture à vous-même.

Ecrire un enregistrement

Nous supposons dans la suite qu'un seul lecteur de disquettes est utilisé.
Prenons un fichier de numéros de téléphone et réduisons-le à l'essentiel: la liste des noms et des numéros correspondants.
Nous utiliserons un enregistrement par personne formé des deux chaînes de caractères nom et numéro (le numéro n'est pas un nombre: d'abord il comporte des blancs, parfois des parenthèses, et surtout il ne peut pas servir à effectuer des calculs!).
Donnons d'abord un nom au fichier: "ANNUAIRE".
Choisissons ensuite un numéro de "canal" pour transférer le fichier sur la disquette. En effet les données situées en mémoire centrale passent d'abord par une sorte de sas ou "zone tampon" (en anglais *buffer*) avant d'être transférées sur la disquette.

Ecriture d'un enregistrement



Prenons le canal n° 1.
Voici le programme qui écrit le premier enregistrement (JULES 000 00 01) du fichier "ANNUAIRE", en passant par le canal n° 1;

```
10 ' ECRITURE ANNUAIRE
20 OPEN "O",#1,"ANNUAIRE"
30 WRITE #1,"JULES","000 00 01"
40 CLOSE #1
```

La ligne 20 ouvre (*OPEN* = ouvrir) la porte du canal n° 1 (#1) qui sort ("O" pour "*OUTPUT*" = sortir) de la mémoire centrale pour aller sur la disquette. Attention à bien mettre la lettre "O" et non pas le chiffre "0".
La ligne 30 écrit (*WRITE* = écrire) le premier enregistrement (c'est-à-dire "JULES 000 00 01") dans la zone tampon n° 1.
La ligne 40 ferme (*CLOSE* = fermer) la communication entre la mémoire centrale et le canal n° 1 et envoie sur la disquette le contenu de la zone tampon n° 1. Plus brièvement, on dit que le fichier est alors fermé. Exécutez ce programme d'écriture:

```
RUN
```

Le lecteur de disquettes (il est supposé seul) se met en marche, lampe verte allumée. Rien ne se passe à l'écran, si ce n'est le OK habituel. Pour constater qu'il s'est écrit quelque chose sur la disquette, demandez le catalogue:

```
DIR
```

Après le programme que vous avez déjà enregistré, vous devez voir:

```
ANNUAIRE.DAT D A 2
```

Le fichier est bien inscrit sur la disquette. Au nom que nous lui avons donné "ANNUAIRE", a été rajouté le suffixe ",DAT" (*DATA* = données). Il en sera toujours ainsi pour les fichiers de données, sauf si vous donnez vous-même un autre suffixe. Comme pour les programmes, le nom de fichier comporte au maximum huit caractères sans virgule ni point.
La lettre "D" précise le type du fichier: Données.
La lettre "A" précise que les données du fichier sont enregistrées en code ASCII.
Le chiffre "2" indique le nombre de "k-octets" occupés par le fichier.
Vous pouvez maintenant enregistrer le programme d'écriture:

```
SAVE "ECRANN"
```

Vérifiez l'opération avec DIR.

Lire l'enregistrement qu'on vient d'écrire

Pour lire le fichier (réduit ici à un seul enregistrement), il faut:
1. Ouvrir la communication entre la mémoire centrale et la disquette par l'intermédiaire d'un canal qui n'est pas nécessairement le même que celui de l'écriture.
2. Transférer en mémoire centrale le contenu du fichier et éventuellement l'afficher à l'écran.
3. Fermer la communication entre la mémoire centrale et la disquette.
Ce qui donne:

```
NEW
10 'LECTURE ANNUAIRE
20 OPEN "I",#1,"ANNUAIRE"
30 INPUT #1,NOM$,TEL$
40 PRINT NOM$,TEL$
100 CLOSE #1
```

En ligne 20 l'ouverture se fait dans le sens de la lecture("I" = INPUT) en mémoire centrale. Nous avons pris le même canal n° 1.
En ligne 30 s'effectue le transfert des données dans deux variables NOM$ et TEL$ en mémoire centrale. La fonction INPUT# joue le même rôle que la fonction INPUT, mais à partir de la disquette ou plus généralement d'un canal, au lieu du clavier.
La ligne 40 vous permet de "voir" le contenu de ces deux variables.
La ligne 100, enfin, ferme la communication avec le canal n° 1.
Exécutez ce programme de lecture:

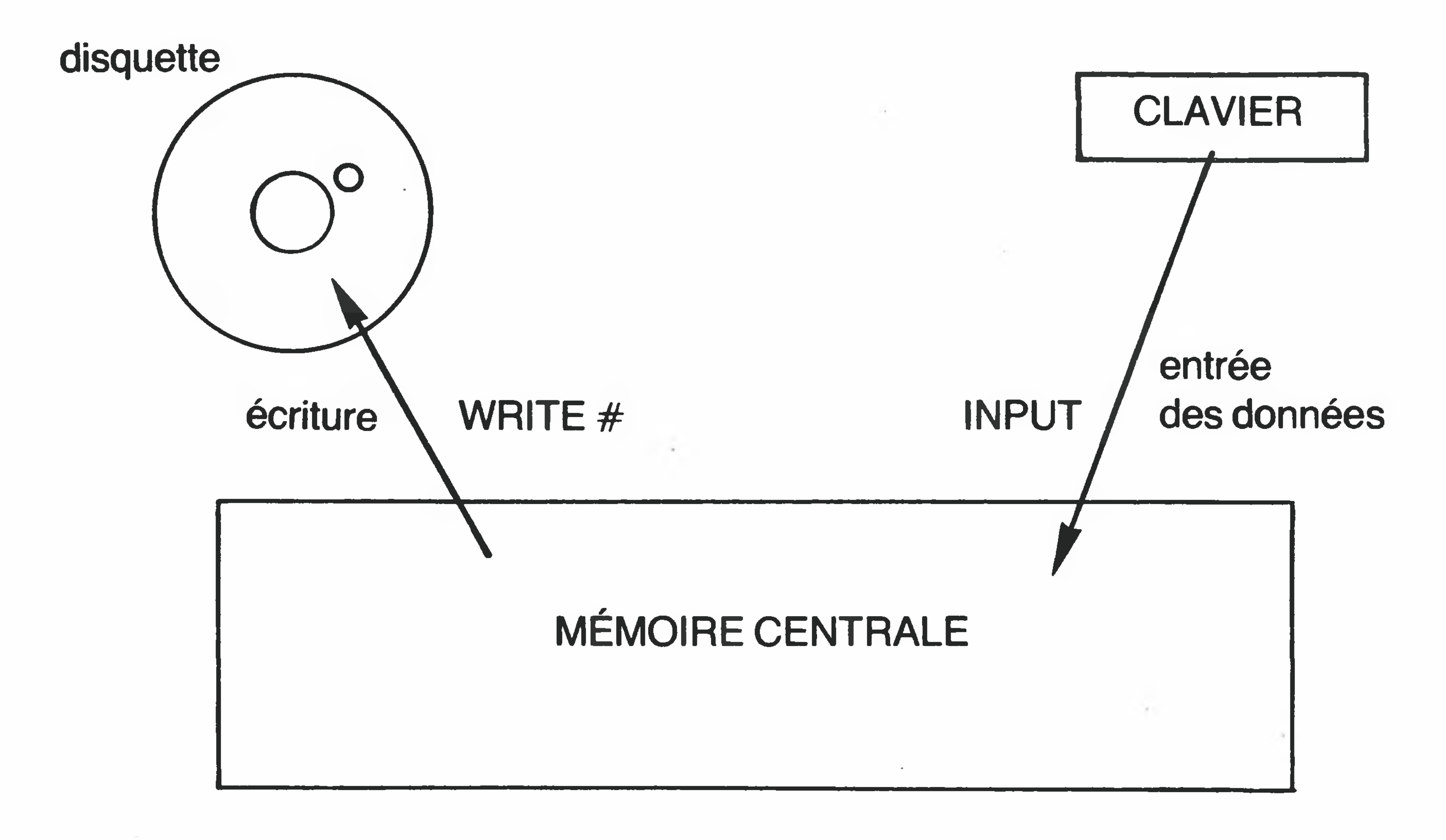
```
RUN
```

Sauvegardez ce programme:

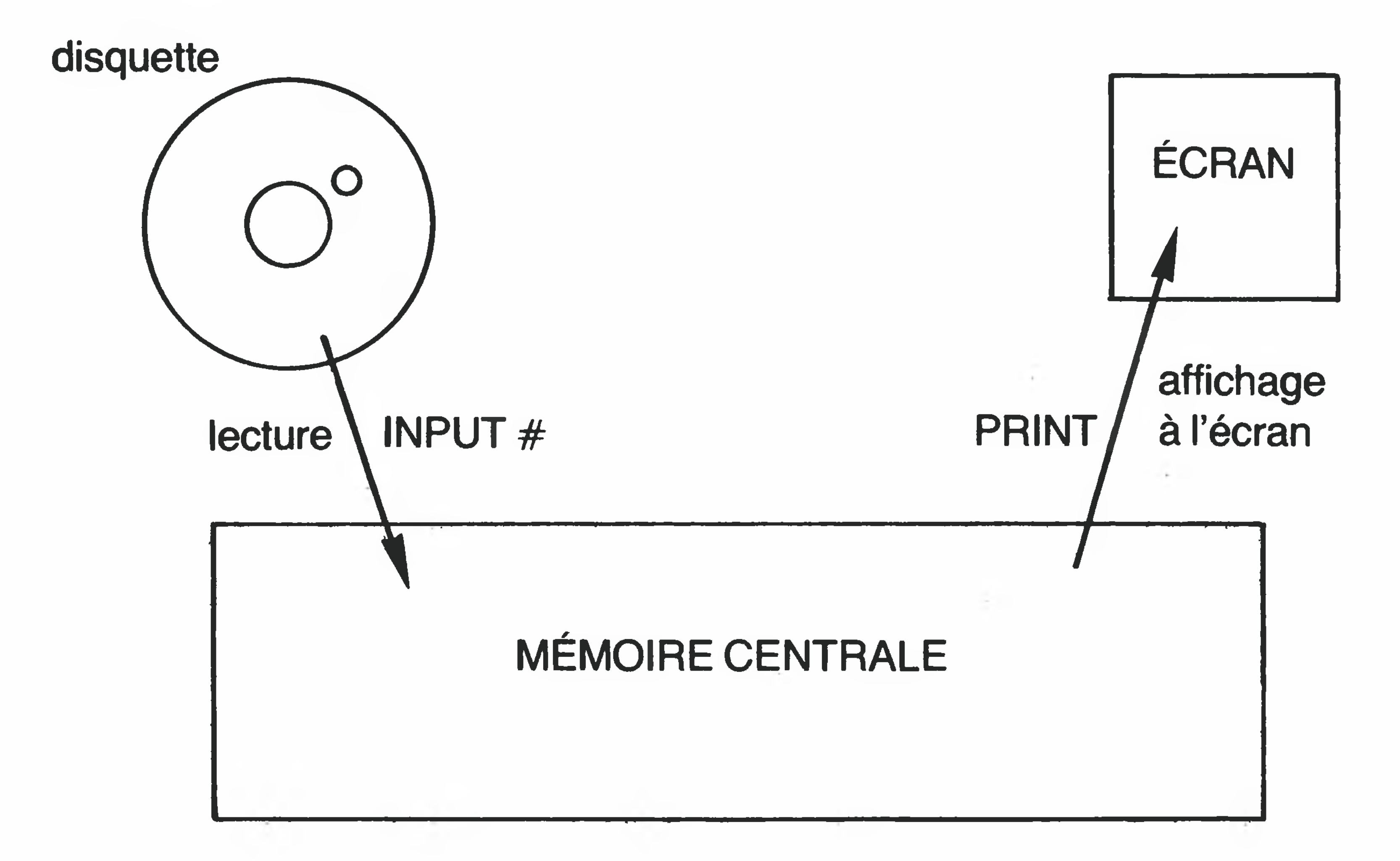
```
SAVE "LECANN"
```

Le schéma qui suit illustre le trajet des données au cours de l'écriture et de la lecture d'un fichier par un programme en mémoire centrale.

Entrée des données et écriture



Lecture et affichage des données



Notez bien que c'est le programme en mémoire centrale qui "lit" ou qui "écrit" le fichier et non pas vous.

Ecrire et lire plusieurs enregistrements

Pour inscrire un deuxième nom dans le fichier, chargeons le programme d'écriture en mémoire et rajoutons une ligne d'écriture:

```
LOAD "ECRANN"
35 WRITE #1,"JIM","000 00 02"
```

Chaque instruction WRITE# définit un nouvel enregistrement. En faisant tourner ce programme modifié, on écrit la nouvelle version du fichier "ANNUAIRE".

```
RUN
SAVE "ECRANN"
```

Cette nouvelle version a effacé l'ancienne: si vous demandez le catalogue, il ne comporte qu'un seul fichier "ANNUAIRE", le dernier écrit. Pour lire le fichier, il faut modifier en conséquence le programme de lecture:

```
LOAD "LECANN"
```

Ajoutons les deux lignes qui correspondent à l'entrée et l'affichage d'un nouvel enregistrement:

```
50 INPUT #1,NOM$,TEL$
60 PRINT NOM$,TEL$
```

La nouvelle version du fichier s'affiche à l'écran:

```
JULES 000 00 01
JIM 000 00 02
```

Si vous voulez éviter d'avoir à enregistrer et à charger en alternance le programme d'écriture et le programme de lecture (qui jouent pour un fichier de données le même rôle que les instructions SAVE et LOAD pour un fichier programme), vous pouvez les regrouper dans un seul programme.

Il est possible de fusionner les deux programmes grâce à l'instruction MERGE:
— renumérotons le programme "LECANN" à partir de la ligne 100 afin qu'il n'y ait pas de lignes communes aux deux programmes.

```
RENUM 100
```

Puis enregistrons-le sous forme ASCII:

```
SAVE "LECANN",A
```

— chargeons le programme "ECRANN" en mémoire centrale:

```
LOAD "ECRANN"
MERGE "LECANN"
LIST
```

Pour pouvoir exécuter séparément le programme d'écriture, il faut ajouter une instruction END en ligne 50.

```
10 ' ECRITURE ANNUAIRE
20 OPEN "O",#1,"ANNUAIRE"
30 WRITE #1,"JULES","000 00 01"
35 WRITE #1,JIM","000 00 02"
40 CLOSE #1
50 END
100 ' LECTURE ANNUAIRE
110 OPEN "I",#1,"ANNUAIRE"
120 INPUT #1,NOM$,TEL$
130 PRINT NOM$,TEL$
140 INPUT #1,NOM$,TEL$
150 PRINT NOM$,TEL$
160 CLOSE #1
```

Pour exécuter la partie écriture, il suffit de taper:

```
RUN
```

Et pour exécuter la partie lecture, il faut taper:

```
RUN 100
```

Vous pouvez sauvegarder ce programme de gestion:

```
5' ECRITURE LECTURE
SAVE "EL-ANN"
```

Un nombre quelconque d'enregistrements

Il est évident qu'avec le procédé précédent (une ligne pour chaque instruction WRITE en écriture et deux lignes pour lire chaque enregistrement), il serait fastidieux de rentrer un fichier important.

Pour répéter la même opération nous ferons donc une itération, c'est-à-dire une boucle, ou plutôt deux boucles: une "en écriture" et une "en lecture".

```
NEW
5 ' ECRITURE LECTURE
7 CLS
10 ' ECRITURE ANNUAIRE
15 PRINT "ENTREE DES NOMS DANS LE FICHIER"
20 OPEN "O",#1,"ANNUAIRE"
40 DO
50   INPUT "NOM:";NOM$
60   IF NOM$="" THEN EXIT
70   INPUT "TELEPHONE:";TEL$
80   WRITE #1,NOM$,TEL$
90 LOOP
100 CLOSE #1
110 PRINT "ECRITURE TERMINEE"
120 END
```

Les lignes 50 à 80 constituent une boucle d'écriture des enregistrements. La boucle est terminée lorsqu'on entre un nom vide, c'est-à-dire lorsqu'on appuie directement sur ENTREE en réponse à la question posée ligne 50.
Essayez le programme:

```
RUN
```

Et répondez avec une liste de noms et de numéros de téléphone.
Voici maintenant le programme de lecture avec sa boucle:

```
200 ' LECTURE ANNUAIRE
210 OPEN "I",#1,"ANNUAIRE"
220 DO
230   INPUT #1,NOM$,TEL$
240   PRINT NOM$,TEL$
250 LOOP
260 CLOSE #1
```

Faites-le tourner:

```
RUN 200
```

S'affichent à l'écran les noms et numéros que vous venez d'écrire.
Mais à la fin vous obtenez le message d'erreur:

```
Input Past End in 230
```

c'est-à-dire lecture après la fin du fichier.
En effet, il n'y a pas de test pour arrêter la boucle: lorsque le dernier enregistrement a été transféré en mémoire centrale, l'instruction INPUT# ne trouve plus rien, d'où le message d'erreur.
Une fonction permet de savoir si un fichier est terminé: c'est la fonction EOF (*End Of File* = fin de fichier).
Tant que le fichier qui transite par le canal n° 1 n'est pas terminé, EOF(1) vaut 0 soit l'état "FAUX". Lorsqu'on atteint la fin du fichier, alors EOF(1) vaut −1, soit l'état "VRAI".
Le test s'écrit donc:

```
225 IF EOF(1) THEN EXIT
```

```
RUN 200
```

Cette fois la lecture du fichier se termine sans message d'erreur.

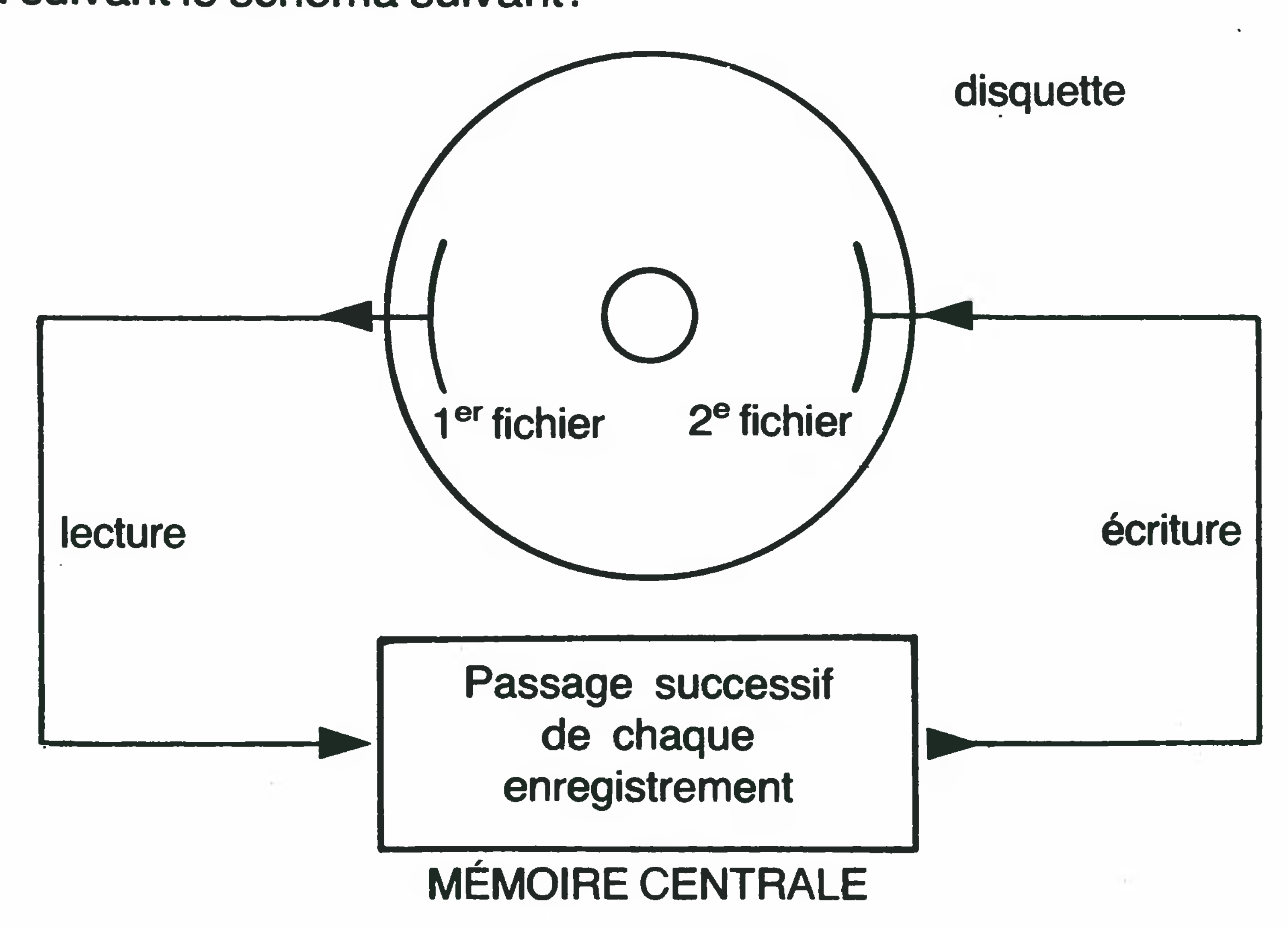
Modifier un enregistrement

D'après ce que nous venons de voir, avec un fichier séquentiel vous êtes ou bien en écriture ou bien en lecture: il est impossible d'être à la fois en "sortie" ("O") et en "entrée" ("I"). La communication entre la disquette et la mémoire centrale est à sens unique (il n'en sera pas de même avec les fichiers à accès direct).

Comment faire pour modifier une donnée au milieu du fichier?
La seule solution est de créer un deuxième fichier qui sera identique au premier sauf pour les enregistrements à modifier. Le fichier original sera transféré en mémoire centrale à travers un canal (n° 1 par exemple) et recopié sur le deuxième fichier à travers un autre canal (n° 2 par exemple). Seules les données à modifier ne seront pas recopiées du premier sur le deuxième fichier: chaque numéro de téléphone modifié sera entré en mémoire centrale par le clavier puis écrit dans le deuxième fichier.

Voici la structure du programme de modification:
— Ouvrir le fichier "ANNUAIRE" en lecture sur le canal n° 1.
— Ouvrir un nouveau fichier "FICNOUV" en écriture sur le canal n° 2.

— Pour tous les enregistrements :
● lire l'enregistrement NOM$, TEL$ (si le fichier 1 n'est pas terminé) et afficher son contenu à l'écran,
● demander si cet enregistrement doit être modifié, si oui entrer la modification,
● écrire l'enregistrement sur le nouveau fichier.
— fermer le canal n° 2 en écriture.
— fermer le canal n° 1 en lecture.
Ou suivant le schéma suivant :



Il est évident que les ronds symbolisant chaque fichier sur la disquette ne représentent pas la surface réelle de mémoire occupée !

```
10 ' MODIFICATION ANNUAIRE
20 OPEN "I",#1,"ANNUAIRE"
30 OPEN "O",#2,"FICNOUV"
40 DO
50   IF EOF(1) THEN EXIT
60   INPUT #1,NOM$,TEL$
70   PRINT NOM$,TEL$
80   INPUT "MODIFICATION (O/N)";REP$
90   IF REP$="O" THEN INPUT "NOUVEAU NUMERO:";TEL$
100   WRITE #2,NOM$,TEL$
110 LOOP
120 CLOSE #1,#2
```

Remarquez en ligne 120 la fermeture des deux canaux en une seule instruction. Vous pouvez même écrire:

```
120 CLOSE
```

ce qui ferme tous les fichiers ouverts.
Essayez le programme: vous pouvez modifier tous les numéros de téléphone que vous voulez.
Lorsque vous avez terminé de parcourir votre annuaire, vous avez deux fichiers, l'ancien et le nouveau.
Vérifiez-le sur le catalogue de la disquette. Il suffit d'éliminer l'ancien et de rebaptiser ANNUAIRE la nouvelle version:

```
KILL "ANNUAIRE.DAT"
NAME "FICNOUV.DAT" AS "ANNUAIRE.DAT"
```

(Dans les instructions KILL et NAME, les suffixes sont indispensables.)

Vérifiez le résultat de l'opération sur le catalogue.
Et enfin sauvegardez votre programme de modification:

```
SAVE "MODANN1"
```

Si vous n'avez qu'un seul numéro à modifier, il est un peu long de parcourir tout l'annuaire. Vous pouvez modifier le programme afin qu'il demande le nom correspondant au numéro à modifier:

```
10 ' MODIFICATION ANNUAIRE version 2
14 INPUT"NOM A MODIFIER";N$
16 INPUT"NOUVEAU NUMERO";T$
70 IF NOM$=N$ THEN TEL$=T$: PRINT "MODIFICATION FAITE "
```

et supprimer les lignes 80 et 90.

```
LIST
RUN
```

Si le message "MODIFICATION FAITE" n'apparaît pas, c'est que le nom N$ n'est pas dans l'annuaire. Sauvegardez cette deuxième version de votre programme de modification:

```
SAVE "MODANN2"
```

En vous souvenant qu'il n'est pas possible à la fois de lire et écrire dans un fichier séquentiel, vous comprenez comment on peut

modifier un fichier séquentiel: en le réécrivant. Pour supprimer une partie, on recopie tout, sauf cette partie.
Pour ajouter de nouveaux noms, on recopie tout et on ajoute à la fin les noms avec leurs numéros de téléphone.

Remarque: n'arrêtez pas le programme de modification au milieu de l'exécution; votre annuaire ne serait qu'à moitié recopié. Si tel était le cas, il vous faudrait tout reprendre depuis le début avant de faire KILL puis NAME.

Un fichier de nombres: les points d'un dessin

Les coordonnées des points d'un dessin peuvent être placées dans une instruction DATA. Mais en les mettant dans un fichier, on peut les appeler dans n'importe quel programme, et utiliser ce dessin sans avoir à réécrire la liste des coordonnées.
Jusqu'à présent les données étaient des chaînes de caractères. Le procédé est le même pour écrire un fichier de nombres:

```
10 ' EC-DESS
20 OPEN "O",#1,"DESSIN"
30 DO
40   INPUT "COL,LIG";COL,LIG
50   IF COL=-1 THEN EXIT
60   WRITE #1,COL,LIG
70 LOOP
80 CLOSE #1

RUN
```

Lorsque les coordonnées du dernier point ont été écrites, donnez à COL la valeur −1 (et n'importe quelle valeur à LIG) pour arrêter l'écriture du fichier.

```
SAVE "EC-DESS"
```

A la lecture, le premier point fixé par PSET (C,L) est celui où doit commencer le dessin:

```
10 ' LE-DESS
20 CLS
30 OPEN "I",#1,"DESSIN"
40 INPUT #1,C,L
50 PSET (C,L)
60 DO
70   IF EOF(1)=-1 THEN EXIT
80   INPUT #1,COL,LIG
90   LINE -(COL,LIG)
100  LOOP
110  CLOSE

RUN
```

Le programme de lecture n'affiche pas les valeurs des variables mais les utilise directement, ici dans une instruction LINE.

```
SAVE "LE-DESS"
```

Les fichiers de nombres s'écrivent et se lisent exactement comme les fichiers de chaînes de caractères: avec WRITE# pour l'écriture et INPUT# pour la lecture.
Vous pouvez donc mélanger des données chaînes de caractères et des données numériques: il suffit que les variables qui suivent une instruction INPUT# correspondent bien aux variables qui suivent l'instruction WRITE# correspondante.
Par exemple:

WRITE#1, NOM$, PRENOM$, AGE **(en écriture)**
INPUT#1, NOM$, PRENOM$, AGE **(en lecture)**

Utiliser plusieurs fichiers de données de même nature

Si vous avez dix dessins différents stockés chacun dans un fichier, il vous faudra a priori dix programmes d'écriture et dix programmes de lecture correspondants (éventuellement encore dix programmes de modification). Or dans ces programmes, la seule chose qui change est le nom du fichier, c'est-à-dire du dessin.
Le bon réflexe en programmation est de toujours placer dans des variables "ce qui change".

Modifions donc "EC-DESS" pour en faire l'unique programme destiné à l'écriture de tous les dessins:

```
10 ' EC-DESS
15 INPUT "NOM DU DESSIN ";DESSIN$
20 OPEN "O",#1,DESSIN$
```

et de même pour le programme de lecture:

```
10 ' LE-DESS
25 INPUT "NOM DU DESSIN ";DESSIN$
30 OPEN "I",#1,DESSIN$
```

De l'importance de la fermeture des fichiers (UNLOAD)

Au cours de l'écriture de fichiers vous avez peut-être constaté que le lecteur ne tourne pas lors de l'entrée de chaque enregistrement. Si vous écrivez peu de données, le moteur démarre seulement après l'entrée de la dernière donnée. Si vous écrivez un grand nombre de données, le moteur démarre de temps en temps. (Remplissez un fichier de 200 ou 300 zéros pour le vérifier!)
En effet, l'ordinateur écrit les données dans une zone tampon (buffer) et ne transfère ces données sur la disquette que lorsque la zone tampon est pleine (ou lorsqu'il reçoit l'ordre CLOSE).
La capacité d'une zone tampon étant d'un secteur (128 octets en simple densité ou 255 en double), le transfert des données de la zone tampon à la disquette se fait donc secteur par secteur.
L'instruction CLOSE a pour objet d'écrire les dernières données de la zone tampon (celle-ci n'étant pas pleine à la fin du fichier) et aussi de noter dans le catalogue de la disquette l'endroit où s'arrête effectivement le fichier.
Cette dernière information est très importante lors de la lecture du fichier. Si vous vous arrêtez au milieu du programme d'écriture à cause d'une erreur, ou parce que vous avez appuyé sur CNT-C, la fermeture du fichier ne sera pas faite.
Si vous ôtez la disquette immédiatement après, l'ordinateur risque de ne plus s'y retrouver: les informations concernant ce fichier (en particulier l'endroit où il se termine) n'auront pas été inscrites sur le catalogue puisque l'instruction CLOSE n'a pas été exécutée. Un autre incident grave risque de se produire si vous mettez alors une autre disquette dans le lecteur. En effet, lors de la prochaine

opération sur fichier, les informations restées en suspens sur la première disquette seront alors écrites sur la seconde, ce qui la brouille ainsi que son catalogue.
Il existe une commande qui vous garantit contre toute erreur de ce genre:
UNLOAD (="décharger").
Si vous tapez:

```
UNLOAD
```

avant de retirer une disquette, vous serez sûr que tous les fichiers seront correctement fermés et toutes les informations les concernant dûment enregistrées sur le catalogue.
A priori cette commande s'applique au lecteur n° 0. Pour qu'elle agisse sur le lecteur n° 1, il suffit d'écrire:

```
UNLOAD 1
```

Recensement général des périphériques utilisables avec un fichier

Vous savez maintenant écrire et lire un fichier sur disquette. D'autres périphériques peuvent être utilisés, mais il faut alors le préciser. En effet en l'absence d'autre indication, lorsque le lecteur de disquettes (n° 0) est branché, il est pris comme le périphérique implicitement souhaité. Si vous voulez utiliser un autre périphérique que le lecteur n° 0, (lecteur n° 1, cassette...)
il faut l'indiquer dans l'instruction d'ouverture du fichier:

en écriture

```
OPEN "O",#1,"1:ANNUAIRE"
```

pour écrire sur le lecteur n° 1,

```
OPEN "O",#1,"CASS:ANNUAIRE"
```

pour écrire sur la cassette,

```
OPEN "O",#1,"LPRT:"
```

pour écrire sur imprimante,

```
OPEN "O",#1,"SCRN:"
```

pour écrire à l'écran,

```
OPEN "O",#1,"COMM:"
```

pour écrire sur la voie de communication série.

L'ensemble "CASS: ANNUAIRE" constitue le descripteur de fichier.

Pour les trois derniers périphériques, il est inutile de préciser le nom du fichier puisqu'il est seulement transmis et non pas conservé dans une mémoire.
De même en lecture:

en lecture

```
OPEN "1",#1,"1 :ANNUAIRE"
```
pour lire à partir du lecteur n° 1,

```
OPEN "1",#1,"CASS: ANNUAIRE"
```
pour lire à partir de la cassette,

```
OPEN "1",#1,"KYBD:"
```
pour lire à partir du clavier (KEYBOARD)

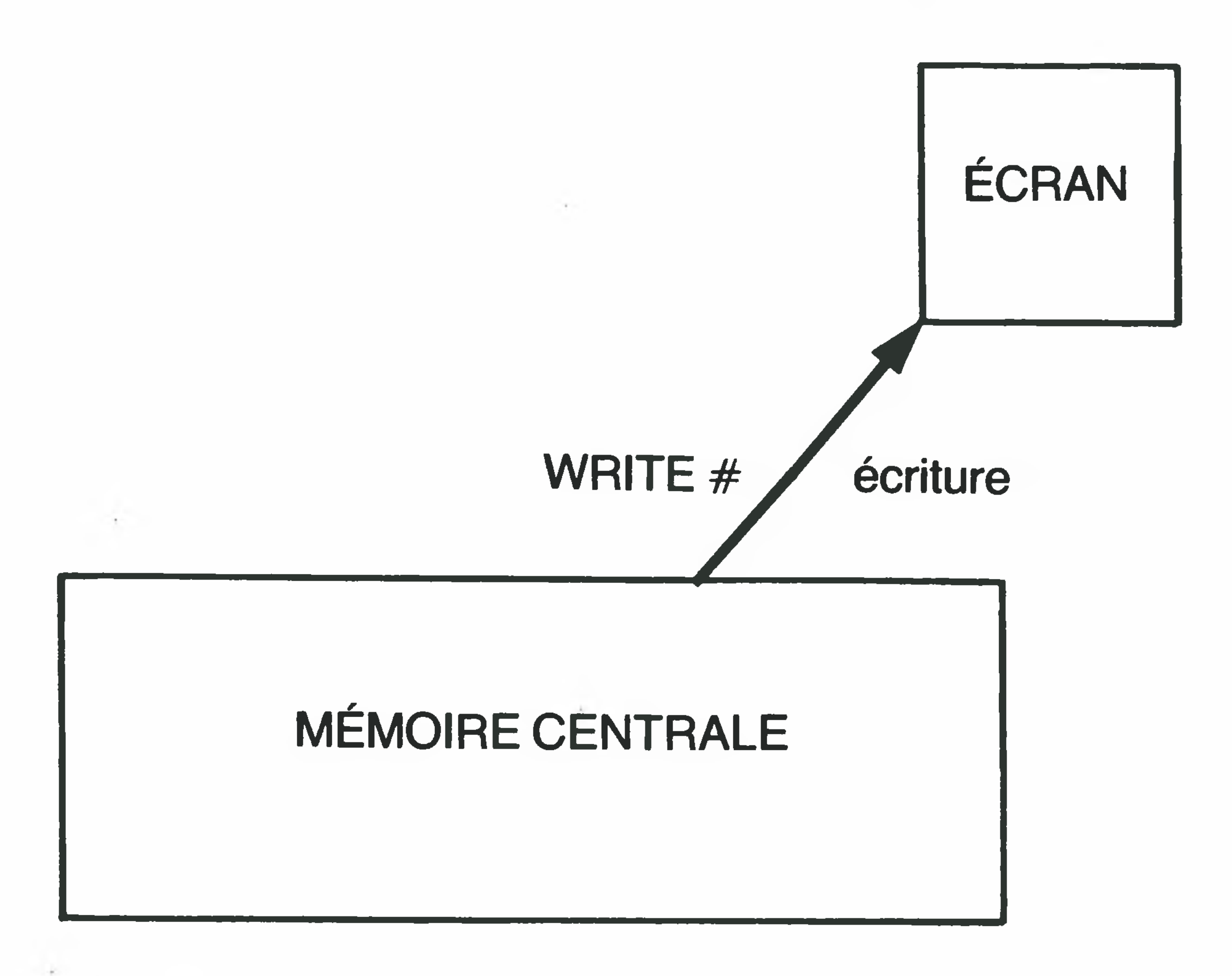
```
OPEN "1",#1,"COMM:"
```
pour lire à partir de la voie série.

Il est évidemment impossible de "lire" un fichier écrit à l'écran ou sur l'imprimante: ces deux périphériques sont exclusivement des périphériques "de sortie".
Seuls les lecteurs de disquettes, de cassettes, et la voie de communication série peuvent être utilisés aussi bien en entrée qu'en sortie.
Deux périphériques sont un peu différents des autres: l'écran et le clavier. En effet, les deux instructions du Basic standard INPUT et PRINT permettent à l'ordinateur de "lire" à partir du clavier (entrée des données) ou d'"écrire" à l'écran (affichage) sans utiliser de canal.
C'est pour mettre ces deux périphériques en parallèle avec les autres que vous est offerte la possibilité d'ouvrir un fichier en lecture avec le clavier (KYBD:) et en écriture avec l'écran (SCRN:).

Voir le fichier s'écrire à l'écran

Jusqu'à présent vous n'avez pas encore vu le fichier lui-même, c'est-à-dire ce qui est effectivement inscrit sur la disquette.
Pour voir un fichier, utilisons l'écran comme périphérique de sortie

en écriture plutôt que le lecteur de disquettes. Inutile de vous dire que ceci ne présente aucun intérêt en pratique car si l'on écrit un fichier, c'est en général pour le garder. Et sur l'écran, la durée de vie du fichier est limitée au déroulement des 25 lignes!



Pour écrire le fichier "ANNUAIRE", chargez en mémoire centrale le programme d'écriture correspondant:

```
LOAD "EL-ANN"
```

Le périphérique implicite est le lecteur de disquettes (n° 0).
Modifions donc la ligne 30:

```
30 OPEN "O",#1,"SCRN: ANNUAIRE"
```

Et pour mieux voir encore le fichier lui-même, écrivons-le en rouge sur fond jaune (en supposant que vous utilisez les couleurs standard bleu sur cyan):

```
75 COLOR 1,3
80 WRITE #1,NOM$,TEL$
85 COLOR 4,6

RUN
```

Dans la sortie sur l'écran, les éléments du fichier s'affichent en rouge sur fond jaune.
Remarquez que l'instruction WRITE écrit les chaînes de caractères entre guillemets. En outre, à la fin de chaque enregistrement elle inscrit les caractères "retour à la ligne" et "saut de ligne" : le curseur

revient au début de la ligne suivante après chaque enregistrement. Cependant sur la disquette, tous les enregistrements sont à la suite les uns des autres, la fin d'un enregistrement étant justement définie par ces caractères "retour à la ligne".

```
SAVE "EL-ANN2"
```

Comparez maintenant avec ce qui se passe pour l'écriture d'un fichier de nombres:

```
LOAD "EC-DESS"

20 OPEN "O",#1,"SCRN: DESSIN"
35 COLOR 3,4
65 COLOR 6,4
```

Les données numériques sont écrites dans le fichier sans guillemets et, comme les chaînes de caractères, séparées par une virgule.

Recopier un fichier sur cassette

Il peut être intéressant de conserver un fichier sur cassette, par exemple pour en garder une copie de sécurité. La gestion d'un fichier sur cassette se fait de la même manière que sur disquette: un programme sert à écrire le fichier (on est alors "en sortie") et un autre programme sert à le lire (on est alors "en entrée").
La seule différence réside dans le "descripteur de fichier" :

```
OPEN "O",#1,"CASS: ANNUAIRE"
```

pour ouvrir le fichier en écriture,

```
OPEN "I",#1,"CASS: ANNUAIRE"
```

pour ouvrir le fichier en lecture.

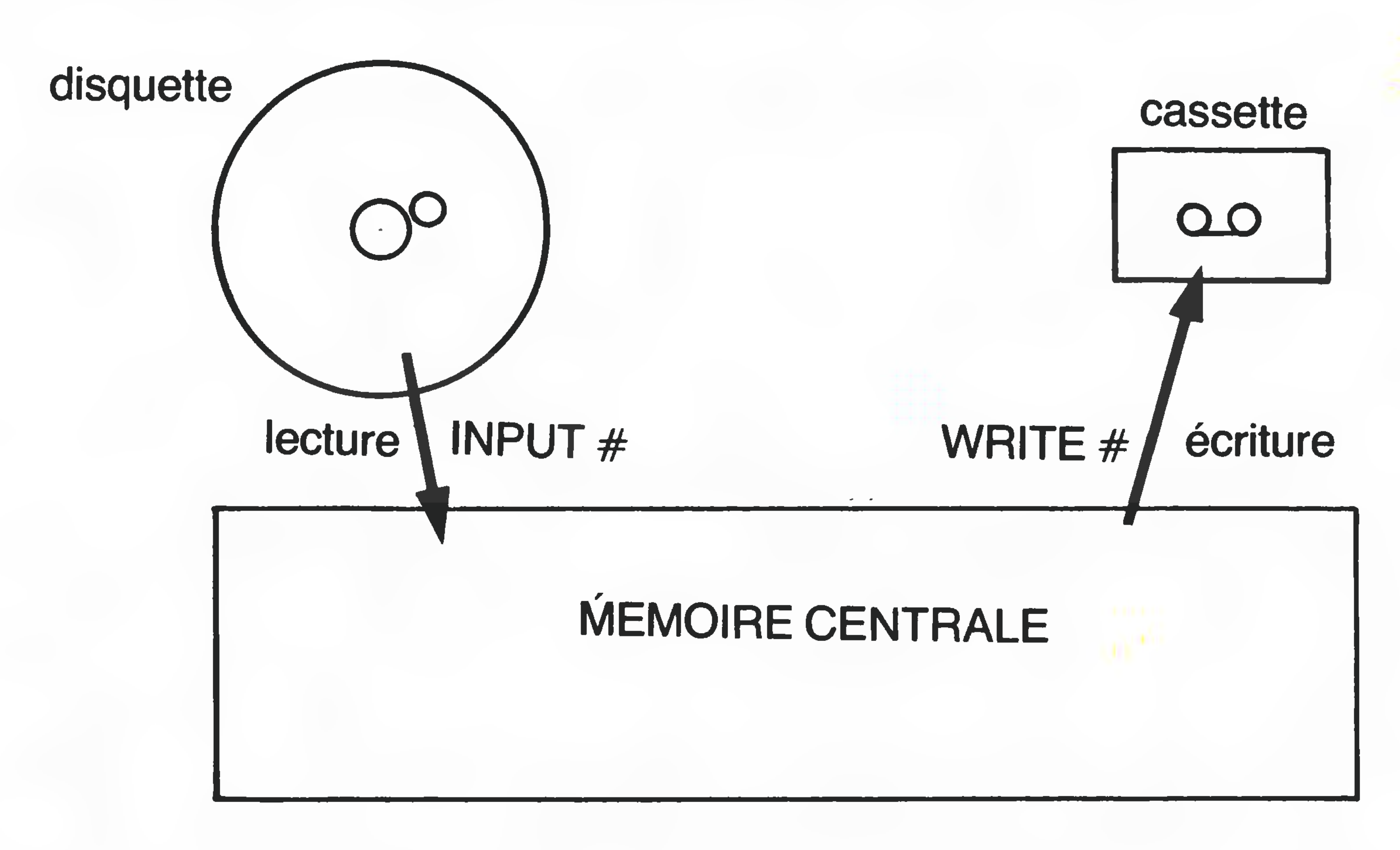
Vous pouvez ainsi écrire le programme qui transfère un fichier d'une disquette (dans le lecteur n° 0) sur une cassette. Il faut:
— ouvrir le fichier sur disquette en lecture (sur un canal)
— ouvrir le fichier sur cassette en écriture (sur un autre canal).

— pour tous les renseignements:
- lire un enregistrement de la disquette,
- écrire cet enregistrement sur la cassette,

— fermer les deux canaux.

```
10 ' DIS-CASS
20 OPEN "I",#1,"ANNUAIRE"
30 OPEN "O",#2,"CASS:ANNUAIRE"
35 DO
40 IF EOF(1) THEN EXIT
50 INPUT #1,NOM$,TEL$
60 WRITE #2,NOM$,TEL$
70 LOOP
80 CLOSE
```

Le schéma du transfert des données est le suivant:



Pour ensuite recopier le fichier de la cassette sur la disquette, il suffit d'inverser les deux descripteurs de fichier dans le schéma et le programme précédents:

```
10 ' CASS-DIS
20 OPEN "I",#1,"CASS:ANNUAIRE"
30 OPEN "O",#2,"ANNUAIRE"
```

Utiliser PRINT#USING pour bien présenter les données sur imprimante

Si vous ne savez pas où s'envole votre argent et si vous avez la volonté et la persévérance nécessaires pour tenir le compte de vos dépenses, vous pouvez noter ces dernières dans un fichier. Vous pourrez ensuite les regrouper, faire des moyennes...
Elaborons d'abord le programme d'écriture sur disquette de ce fichier de dépenses:

```
10 ' EC-DEP
20 OPEN "O",#1,"DEPENSES"
30 WRITE #1,"ESSENCE",160.50
40 WRITE #1,"LIVRES",78.80
50 CLOSE #1
```

Attention: si vous écrivez 160,50 (avec une virgule au lieu d'un point), l'ordinateur écrira deux nombres différents 160 et 50.

```
RUN
```

Voilà, le fichier est écrit.

```
SAVE "ECRDEP"
```

Pour afficher les données à l'écran, il suffit de relire les données (INPUT#) et de les afficher (PRINT):

```
10 ' LE-DEP
20 OPEN "I",#1,"DEPENSES"
30 DO
40   IF EOF(1) THEN EXIT
50   INPUT #1,OBJET$,PRIX
60   PRINT OBJET$,PRIX
70 LOOP
80 CLOSE #1
```

```
RUN
```

Mais peut-être préféreriez-vous que les données s'affichent de manière plus lisible:

```
ESSENCE 160.50 F
LIVRES 78.80 F
```

Pour cela, ajoutez la ligne 25 et modifiez la ligne 60:

```
25 PRE$="%                %####.## F"
60 PRINT USING PRE$;OBJET$,PRIX
```

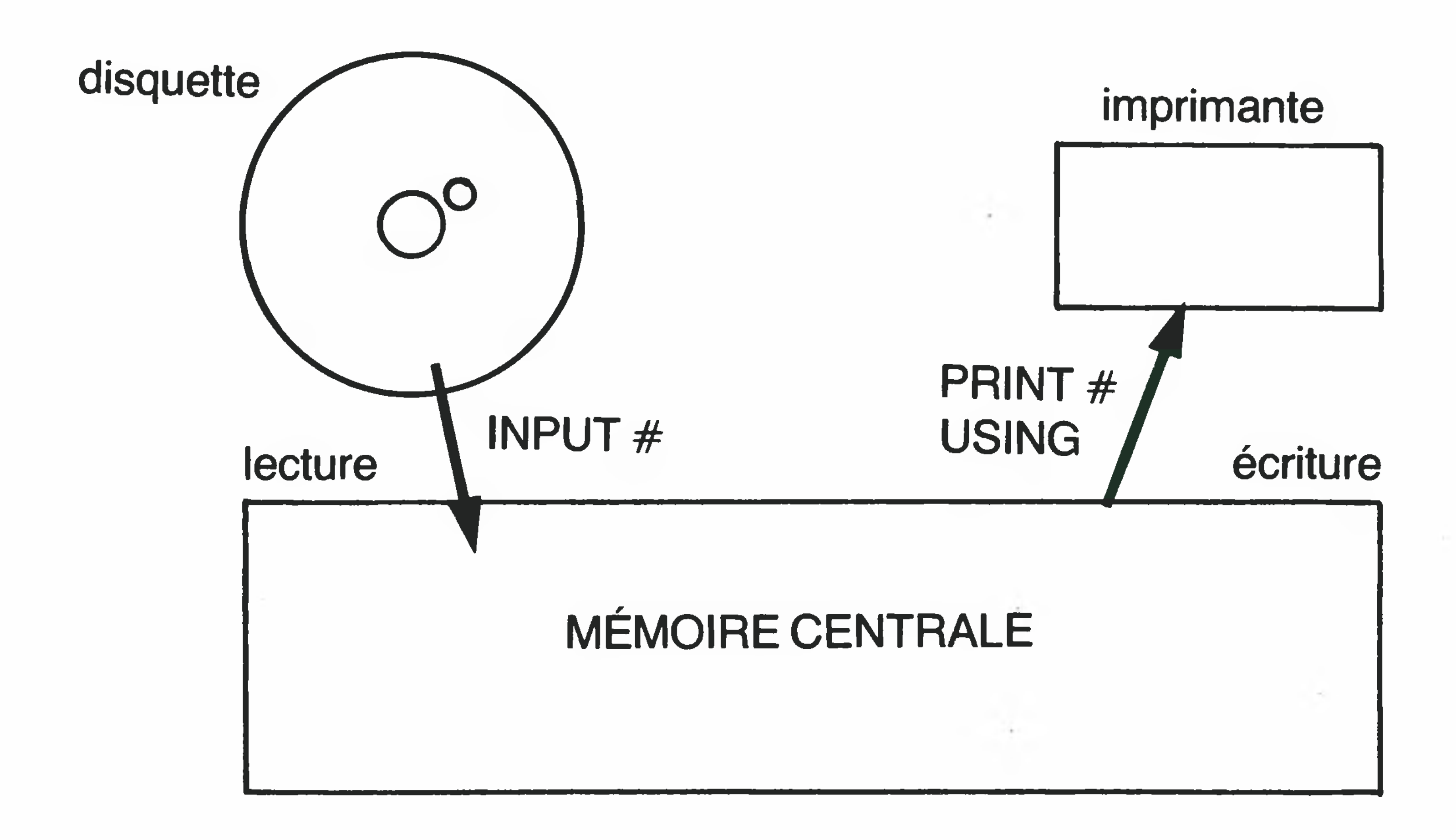
L'instruction PRINT USING permet d'afficher des données suivant une disposition (ou un "format") prévue à l'avance.
Essayez donc avec une chaîne de caractères:

```
PRINT USING"%     %";"EPICERIE"
```

La chaîne est affichée avec le même nombre de caractères que le format en contient, soit trois caractères dans ce cas.
Essayez avec un nombre:

```
PRINT USING "##.##";5.234
```

Le nombre est affiché suivant le modèle du format: deux chiffres avant le point décimal et deux chiffres après.
Essayez avec d'autres nombres.
Pour notre fichier, nous avons même ajouté la lettre "F" à la fin de chaque prix: on peut ajouter des caractères dans le format.
Si vous voulez garder une trace écrite de votre fichier, il va falloir l'écrire sur imprimante et donc ouvrir un fichier en écriture sur l'imprimante.



```
10 ' LE-DEP
20 OPEN "I",#1,"DEPENSES"
22 OPEN "O",#2,"LPRT:"
25 PRE$= "%                    %####.## F
30 DO
40   IF EOF(1) THEN EXIT
50   INPUT #1,OBJET$,PRIX
60   PRINT #2,USING PRE$;OBJET$,PRIX
70 LOOP
80 CLOSE #1,#2
```

Résultat: une belle impression sur papier.
Si l'imprimante n'est pas en marche, le programme se bloque.
Appuyez sur la touche d'initialisation pour retrouver votre programme et sauvegardez-le:

```
SAVE "IMPDEP"
```

Mais maintenant il vous faut allumer l'imprimante, alors attention à l'ordre d'allumage:
— retirez la disquette du lecteur,
— éteignez l'unité centrale puis les autres périphériques,
— branchez l'imprimante et allumez-la,
— rallumez les autres périphériques,
— remettez la disquette dans le lecteur.
Pour éviter toute cette gymnastique, allumez tous les périphériques que vous risquez d'utiliser dès le début de votre travail.

Règle de bon usage

Si vous arrêtez un programme d'écriture de fichier par CNT-C, ou s'il s'arrête par un message d'erreur, fermez tous les fichiers en mode direct, et recopiez toutes les interventions récentes sur la disquette par:

```
UNLOAD
```

Vous pourrez alors retirer la disquette du lecteur sans perdre son contenu.

Fonctions et instructions vues

OPEN
WRITE#
CLOSE
INPUT#
EOF
PRINT#
UNLOAD

Chapitre 14
Les fichiers à accès direct

Les fichiers à accès direct ont été créés pour tirer un meilleur parti des disquettes et des disques.
En effet, dans un fichier séquentiel, pour lire le 100ᵉ enregistrement, il faut commencer au début et lire d'abord les 99 autres avant d'y arriver. Dans un fichier à accès direct, on peut accéder directement au 100ᵉ enregistrement. C'est un peu comme un cahier pour noter des noms. Si vous écrivez tous les noms sur le même cahier les uns à la suite des autres, il vous faut relire tout depuis le début pour arriver à celui que vous cherchez. Si vous notez vos noms dans un répertoire, il suffit par la suite d'aller à la lettre qui vous intéresse pour les retrouver.
L'inconvénient du fichier à accès direct, c'est que chaque case du fichier doit être parfaitement définie à l'avance (comme les découpages dans le répertoire) et qu'en outre chaque enregistrement doit avoir la même longueur.

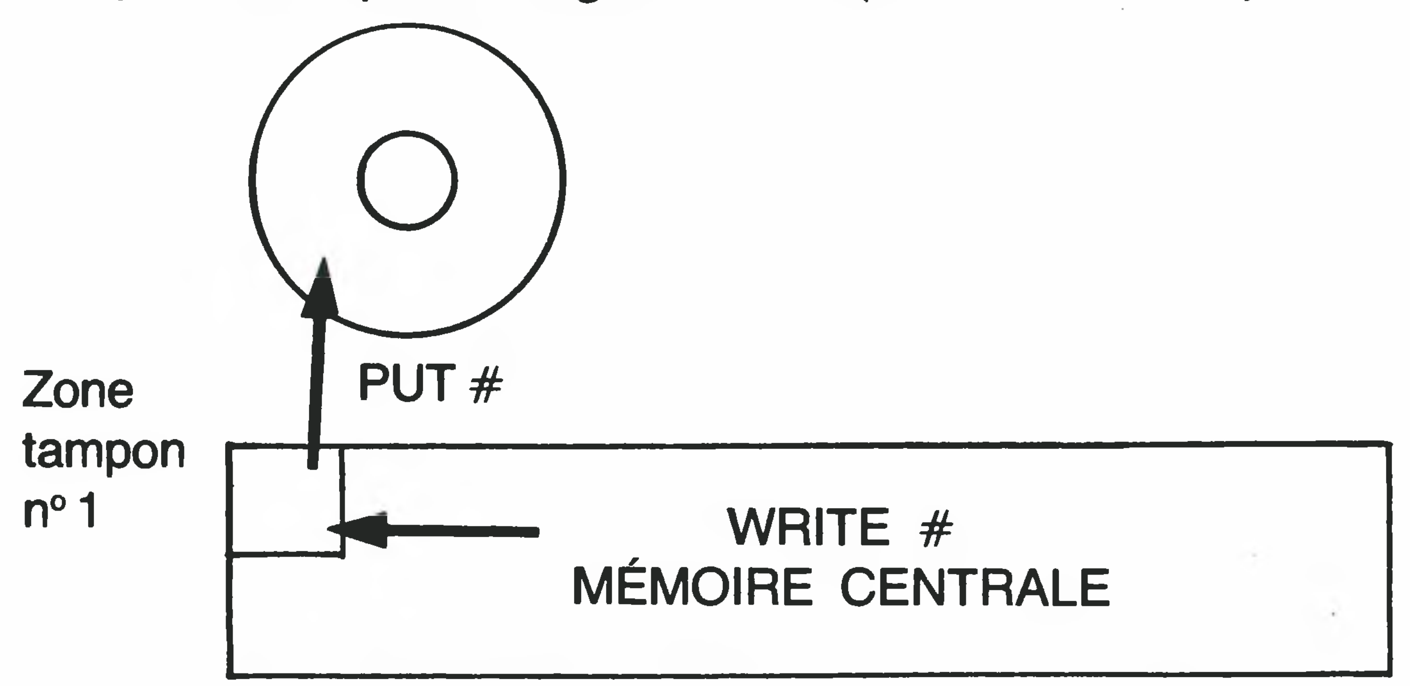
Une chose pour chaque jour, un jour pour chaque chose: un agenda simplifié

Prenons un exemple concret. Nous allons essayer de transformer notre micro-ordinateur en agenda de bureau.

La fonction principale d'un agenda est de pouvoir noter, quand cela se présente, tout ce à quoi il faut penser les jours suivants. Pour simplifier, nous allons nous limiter à un mois entier, c'est-à-dire 31 jours, et nous allons noter un seul événement par jour.
En prenant pour nom du fichier le nom du mois en question ("mai" par exemple), comment faire pour écrire que le 9, il ne faut pas oublier le cours de tennis?
La solution est très proche d'un fichier séquentiel.

```
10'AGENDA ECRITURE
20 OPEN "D",#1,"MAI"
30 WRITE #1,"TENNIS"
40 PUT #1,9
100 CLOSE #1
RUN
```

Comme pour tout fichier, l'écriture d'un enregistrement comporte trois phases: ouverture du fichier (OPEN), l'écriture des données et la fermeture du fichier (CLOSE). La première différence se voit à l'ouverture: OPEN est suivi de "D" (pour *Direct*) ou encore "R" (pour *Random*).
La deuxième différence réside dans l'écriture qui se fait en deux temps. D'abord WRITE# écrit la donnée dans la zone tampon réservée au fichier, puis PUT# transfère le contenu de la zone tampon (qui constitue un enregistrement) sur la disquette. Tous les enregistrements auront donc la même longueur, celle de la zone tampon, soit 128 octets en simple densité et 255 en double.
L'instruction PUT# indique dans quel fichier il faut écrire (ici le canal nº 1) et dans quel enregistrement (ici le neuvième).



Pourquoi cette deuxième instruction pour écrire?
Dans un fichier séquentiel, après WRITE#, le Basic sait où il doit écrire les données dans le fichier: à la suite des précédentes (ou au début pour les premières). Dans un fichier à accès direct, il faut une

instruction pour indiquer où les écrire, puisqu'on a le droit de les écrire n'importe où.
Essayons d'en écrire un peu plus. Le 14, vous avez prévu une randonnée à bicyclette?

```
50 WRITE #1,"RANDO A VELO"
60 PUT #1,14
```

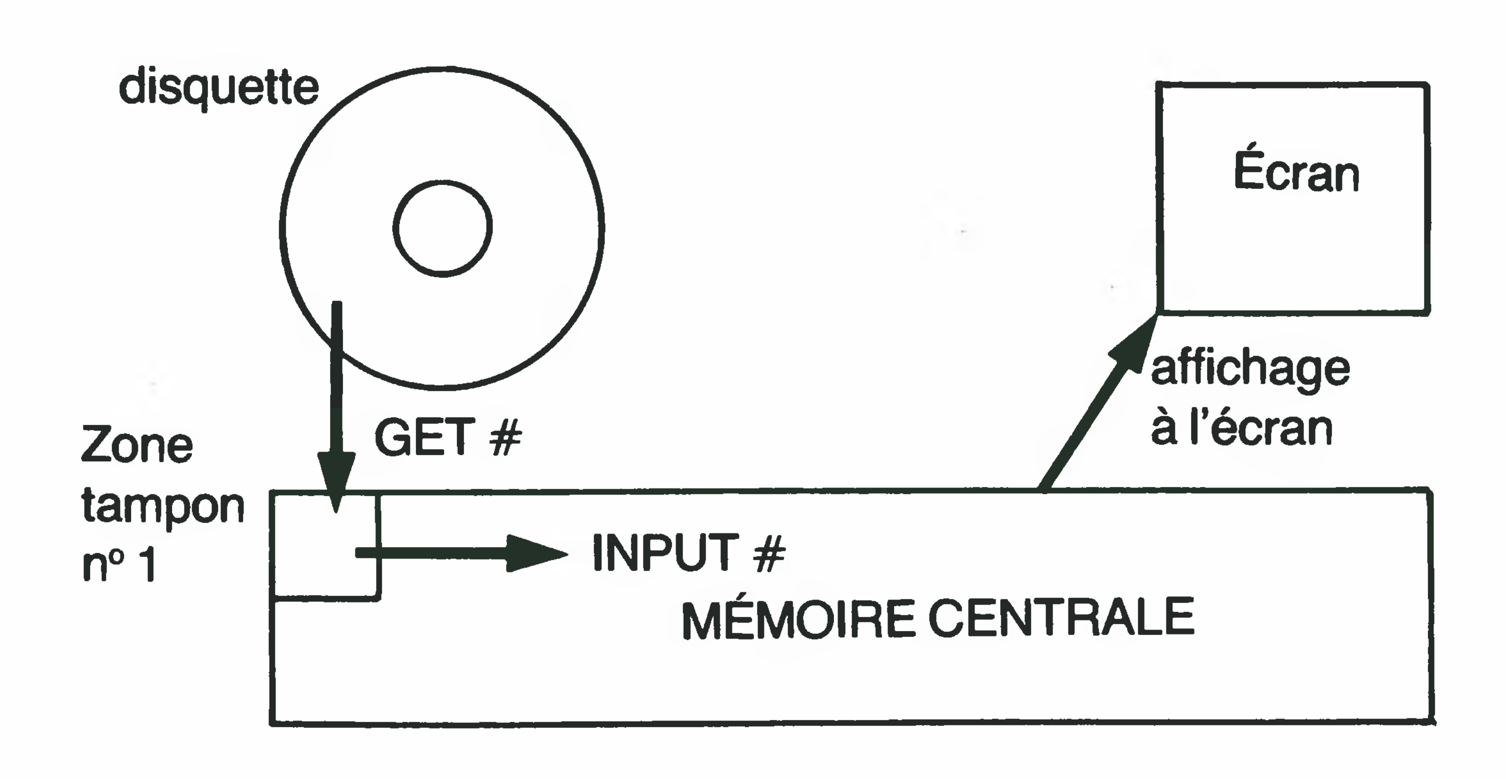
Faites-le tourner:

```
RUN
```

Il est temps maintenant de relire ce qui a été écrit.
En voici le programme:

```
10 ' AGENDA LECTURE
20 OPEN "D",#1,"MAI"
30 GET #1,9
40 INPUT #1,QUOI$
50 PRINT "LE 9 ",QUOI$
60 GET #1,14
70 INPUT #1;QUOI$
80 PRINT "LE 14 ";QUOI$
100 CLOSE #1
```

L'ouverture est identique en lecture et en écriture:
OPEN "D",..., ce qui veut dire qu'un fichier à accès direct est utilisable, sur le même canal, en lecture et en écriture; nous y reviendrons. La lecture des données se fait également en deux temps. D'abord le transfert par GET# de l'enregistrement n° 9 dans la zone tampon n° 1.
Ensuite la lecture des données situées dans cette zone tampon par INPUT# comme pour un fichier séquentiel.



Il faut remarquer tout de suite la correspondance entre PUT# et GET# d'une part et entre WRITE# et INPUT# d'autre part. D'ailleurs, pour le même enregistrement, les données lues par INPUT# doivent correspondre aux données écrites avec WRITE#. Dans cet exemple, il n'y a qu'une donnée, c'est facile. Mais si vous voulez écrire des nombres et des chaînes de caractères avec WRITE# dans l'enregistrement, il faudra les relire dans le même ordre avec INPUT# (problème que nous avons déjà rencontré avec les fichiers séquentiels).

Il serait intéressant de pouvoir consulter notre fichier agenda pour n'importe quel jour du mois. Reprenons pour cela notre programme de lecture. Comment le modifier pour qu'il vous demande "QUEL JOUR?" et affiche à l'écran le contenu de l'enregistrement correspondant à ce jour?

```
30 INPUT"QUEL JOUR";J
50 GET #1,J
60 INPUT #1,QUOI$
70 PRINT J,QUOI$
```

Et pour consulter plusieurs jours ajoutez ces deux lignes:

```
25 DO
40 IF J=0 THEN EXIT
80 LOOP
```

Essayez ce nouveau programme. Tant que vous n'indiquez pas un nombre nul, vous obtenez le contenu de l'agenda ou plutôt, le contenu de ce qui a été écrit pour les jours 9 et 14 et des choses bizarres pour les autres jours. Prenez par exemple le jour 3; résultat environ trois lignes de caractères quelconques.
En demandant ainsi la lecture d'enregistrements qui n'ont pas été écrits, vous pouvez obtenir des effets curieux: la touche RAZ n'efface plus la totalité de l'écran, le défilement des lignes à l'écran ne se fait plus normalement... Une seule solution: appuyez sur la touche INITIALISATION, et reprenez votre programme.
Pour le jour 25, vous obtenez "Input past End in 50"

Apparemment, pour 25, nous avons essayé de lire au-delà de la fin du fichier.

Il existe une fonction qui va nous éviter de lire au-delà du dernier enregistrement connu. LOF(1) donne le numéro du dernier enregistrement du fichier numéro 1 (*Last Of File* = dernier du fichier). Vous pouvez ajouter:

```
35 IF J > LOF(1) THEN 30
```

```
10 ' AGENDA LECTURE
20 OPEN "D",#1,"MAI"
25 DO
30   INPUT"QUEL JOUR";J
35   IF J> LOF(1) THEN 30
40   IF J=0 THEN EXIT
50   GET #1,J
60   INPUT#1,QUOI$
70   PRINT J,QUOI$
80 LOOP
100 CLOSE #1
```

Cela ira déjà mieux. Mais cette modification ne nous satisfait pas totalement. La meilleure solution va être d'initialiser avant usage le fichier agenda.
Sauvegardez le programme de lecture et écrivez ce programme d'initialisation du fichier. Il va remplir à l'avance tous les enregistrements du fichier par des espaces pour ne pas récupérer de choses bizarres quand il n'y a rien dans l'agenda pour un jour précis.

Si nous sommes toujours à préparer l'agenda du mois de mai, il y a 31 jours donc 31 enregistrements. Nous allons donc remplir chaque enregistrement par des espaces. Rappelons qu'a priori, la longueur d'un enregistrement est de 128 octets (ou 255). Allez-y.
Voici notre propre solution:

```
10 INI-AGEN

20 OPEN "D",#1,"MAI"
30 FOR J=1 TO 31
40   WRITE #1,SPACE$(255)
50   PUT#1,J
60 NEXT J
70 CLOSE #1
```

L'exécution est à peine lancée que nous obtenons le message: End of Record in 40.
Eh oui! Nous avions oublié que l'instruction WRITE# écrit les chaînes de caractères entre guillemets et ajoute à la fin les deux caractères: "saut de ligne" et "retour en début de ligne". Vouloir écrire 259 caractères dans un enregistrement de 255 a déclenché une erreur. Il suffit de corriger la ligne 40:

```
40 WRITE #1,SPACE$(251)
```

Cette fois, vous pouvez reprendre le programme de lecture. Immédiatement après l'initialisation, pour chaque jour demandé, vous obtenez un peu plus de trois lignes d'espaces. C'est correct. L'initialisation d'un fichier à accès direct n'est pas obligatoire. Mais si vous êtes amené à consulter des enregistrements dans lesquels vous n'avez pas écrit, elle est bien préférable. De plus, comme tous les enregistrements sont initialisés avec la même valeur, vous pouvez facilement déterminer, par programme, si l'un d'eux a reçu des informations. Puisqu'un agenda sert autant à écrire des notes qu'à les relire, regroupons nos deux programmes de lecture et d'écriture en un seul.

Ce programme général AGENDA aura la structure suivante:
1. Ouvrir le fichier du mois.
2. Choisir entre Lecture, Ecriture ou Rien.
3. Si Rien, fermer le fichier et fin du programme.
4. Si Lecture, demander le jour et donner le contenu de l'enregistrement.
5. Si Ecriture, demander le jour, les notes et les écrire dans l'enregistrement.

```
10 ' AGENDA
20 OPEN "D",#1,"MAI"
25 DO
30   INPUT "LECTURE(1),ECRITURE(2) OU RIEN(0)";REP
40   IF REP=0 THEN EXIT
50   ON REP GOSUB 100,200
60 LOOP
70 CLOSE #1
80 END
100 ' LECTURE
110 INPUT "QUEL JOUR";J
120 IF J>LOF(1) THEN RETURN
```

```
130 GET #1,J
140 INPUT #1,QUOI$
150 PRINT "LE";J;QUOI$
160 RETURN
200 ' ECRITURE
210 INPUT "QUEL JOUR";J
220 IF J>LOF(1) THEN RETURN
230 INPUT "NOTER";QUOI$
240 WRITE #1,QUOI$
250 PUT#1,J
260 RETURN
```

Pour éviter l'affichage de quatre lignes blanches lorsqu'un jour est vide, il suffit d'ajouter un message:

```
145 IF QUOI$=SPACE$(251) THEN QUOI$="RIEN"
```

Plusieurs données dans un enregistrement

Il est encore facile d'améliorer l'agenda en notant l'heure correspondant à l'occupation inscrite:

```
235 INPUT "A QUELLE HEURE";H$
240 WRITE #1,QUOI$,H$
```

Il faut modifier en conséquence la lecture:

```
140 INPUT #1,QUOI$,H$
150 PRINT "LE";J;QUOI$;" A";H$
```

Si vous le préférez, vous pouvez noter chaque jour l'occupation du matin et celle de l'après-midi. Il faut seulement veiller à ce que la lecture (INPUT#) des variables soit exactement la même que leur écriture (WRITE#).

Modifier la taille d'un enregistrement

Nous avons vu que la taille d'un enregistrement est celle d'une mémoire tampon (1 secteur). En effet, PUT# et GET# effectuent le transfert de toute la zone tampon entre la disquette et la mémoire centrale. Vous le vérifiez d'ailleurs facilement sur le catalogue: la taille du fichier "MAI" est de 31 secteurs, donc 8 Koctets
Si vous avez peu de chose à écrire dans chaque enregistrement, comme par exemple une seule occupation par jour, cela fait

beaucoup de blancs sur la disquette, et autant de place de perdue. Vous pouvez réduire (ou augmenter) la taille d'un enregistrement. Si par exemple chaque enregistrement ne dépasse pas une ligne à l'écran, soit 40 caractères, il suffit d'ajouter 40 à la fin de l'instruction OPEN (en lecture et en écriture):

```
OPEN "D",#1, "AOUT", 40
```

Attention: impossible de changer de valeur pour un enregistrement: une fois ce nombre fixé, tous les enregistrements sont limités à quarante caractères.

Découper un enregistrement en plusieurs morceaux

Nous allons maintenant voir une deuxième manière (plus performante) de placer plusieurs données dans un enregistrement.
Vous êtes organisateur des 15 km de Trifouillis-les-Bains, grand rassemblement de centaines d'adeptes de la course à pied.
Comment utilser votre micro-ordinateur?

Pour vous, la première étape est d'enregistrer les candidats. Pour chaque candidat, il faut noter son nom, son prénom et son numéro de dossard. Appelons le fichier: "INSCRIPT".
Le numéro de dossard sera le numéro de l'enregistrement sur la disquette. Afin de ne pas perdre de place sur la disquette, limitons la taille d'un enregistrement à quarante caractères:

```
20 OPEN "D",#1,"INSCRIPT",40
```

Une instruction FIELD permet de diviser chaque enregistrement en zones réservées à des variables différentes:

```
30 FIELD #1,15 AS NOM$,15 AS PRENOM$
```

réserve les quinze premiers caractères de chaque enregistrement pour NOM$ et les quinze suivants pour PRENOM$ (*FIELD* = champ). Il reste encore 10 caractères pour inscrire ultérieurement

d'autres données. Pour écrire les deux données NOM$ et PRENOM$, nous utiliserons l'instruction LSET (*Left SET* = placer à gauche) au lieu de WRITE#:

```
10 ' INSCRIPTION ECRITURE
20 OPEN "D",#1,"INSCRIPT",40
30 FIELD #1,15 AS NOM$,15 AS PRENOM$
40 LSET NOM$="DUMOULIN"
50 LSET PRENOM$="HENRI"
60 PUT #1,1
70 CLOSE #1
```

L'instruction LSET place le nom dans la zone de quinze caractères réservée pour NOM$, en le cadrant à gauche de la zone et en remplissant le reste de la zone avec des blancs:

DUMOULIN

Si vous inscrivez M. DUBOUT DE LA MONTAGNE, son nom sera tronqué sans aucun respect par l'ordinateur en:

DUBOUT DE LA MO

C'est à vous de faire preuve de tact!
En passant signalons que l'instruction LSET gagne quatre caractères par rapport à WRITE# (deux pour les guillemets et deux pour le retour à la ligne). Si la donnée à écrire est un nombre, vous pouvez la cadrer à droite de la zone réservée par l'instruction RSET (*Right SET*).

Reprenons maintenant le programme d'écriture pour que:
— il demande le nom et le prénom,

```
35 INPUT"NOM";R$
40 LSET NOM$=R$
45 INPUT"PRENOM";R$
50 LSET PRENOM$=R$
```

— il inscrive tous les coureurs (on arrêtera le programme avec un nom fictif comme “Z”)

```
10 ' INSCRIPTION
20 OPEN"D",#1,"INSCRIPT",40
30 FIELD #1,15 AS NOM$,15 AS PRENOM$
33 I=1
34 DO
35   INPUT"NOM";R$
37   IF R$="Z" THEN EXIT
40   LSET NOM$=R$
45   INPUT"PRENOM";R$
50   LSET PRENOM$=R$
60   PUT #1,1
65   I=I+1
66 LOOP
70 CLOSE #1
```

```
RUN
```

Allez-y, inscrivez vos coureurs. Le test d’arrêt en ligne 37 est indispensable: si vous arrêtez un programme d’écriture de fichier par CNT-C (ou un message d’erreur), le fichier n’est par fermé puisque l’instruction CLOSE n’a pas été exécutée.
Vous risquez ensuite d’écrire involontairement n’importe quoi sur la disquette. Dans ce cas-là, il est prudent de faire:

```
UNLOAD
```

pour fermer les fichiers restés ouverts.

Une facilité d’écriture dans PUT# et GET#

Le remplissage de ce fichier à accès direct se fait de façon séquentielle: vous inscrivez le numéro 1, le 2, le 3...
(Ce n’était pas le cas dans le remplissage de l’agenda).
Dans ce cas vous n’êtes pas obligé de préciser le numéro de l’enregistrement dans l’instruction PUT#: les inscrits sont automatiquement affectés aux enregistrements successifs (le premier étant le n° 1).

Vous pouvez donc supprimer la variable de comptage I : effacez les lignes 33 et 64 et modifiez la ligne suivante :

```
60 PUT#1
```

Pour connaître le nombre d'inscrits, c'est-à-dire le numéro du dernier inscrit, ajoutez :

```
68 PRINT "NOMBRE D'INSCRITS" ; LOF(1)
```

Cette facilité d'écriture dans PUT# existe également avec GET# : si vous lisez les enregistrements les uns après les autres vous n'êtes pas obligé de préciser le numéro de l'enregistrement.

Relire les chaînes de caractères (lire les inscrits)

Si vous voulez savoir qui est inscrit sous le dossard n° 3, il suffit de lire l'enregistrement correspondant :

```
10 ' LECTURE DES INSCRITS
20 OPEN "D",#1,"INSCRIPT",40
30 FIELD #1,15 AS NOM$,15 AS PRENOM$
40 GET #1,3
50 PRINT "NOM:";NOM$
60 PRINT "PRENOM:";PRENOM$
100 CLOSE #1
```

Si vous avez inscrit une centaine de noms, la lecture du 80ᵉ inscrit sera infiniment plus rapide que dans un fichier à accès séquentiel où il faut lire 79 enregistrements avant d'accéder au 80ᵉ.
Inutile dans la lecture d'un enregistrement de faire INPUT# après GET# : le contenu de la variable NOM$ qui a été déposé dans la zone tampon n° 1 par GET# est directement accessible.

Mais attention, une variable de champ définie par une instruction FIELD, comme NOM$, ne peut être modifiée que par une instruction LSET (ou RSET) : elle est toujours à la même place dans la mémoire et a une longueur fixée à l'avance (dans l'instruction FIELD).
Reprenez maintenant le programme de lecture d'un enregistrement comme nous l'avons déjà fait pour le programme d'écriture :

— demander quel est le numéro désiré (N),
— lire l'enregistrement désiré.

On arrête le programme avec un numéro inutilisé, par exemple 0.

```
10 LECTURE DES INSCRITS
20 OPEN "D",#1,"INSCRIPT",40
30 FIELD #1,15 AS NOM$,15 AS PRENOM$
32 DO
40   INPUT "NUMERO ";N
50   IF N=0 THEN EXIT
60   GET #1,N
70   PRINT "NOM:";NOM$
80   PRINT "PRENOM:";PRENOM$
90 LOOP
100 CLOSE #1

RUN
```

Vous relisez les noms et prénoms des coureurs déjà inscrits.

Effacer le contenu d'un enregistrement (supprimer un inscrit)

Revenons à la course qui n'en est qu'à ses préparatifs. Un des candidats déclare forfait avant le départ de la course. Il faut annuler son enregistrement, par exemple en inscrivant quinze blancs pour le nom et le prénom:

```
10 ' SUPPRESSION D'UN INSCRIPT
20 OPEN "D",#1,"INSCRIPT",40
30 FIELD #1,15 AS NOM$,15 AS PRENOM$
40   INPUT "NUMERO A SUPPRIMER";N
50   LSET NOM$=SPACE$(15)
60   LSET PRENOM$=SPACE$(15)
70   PUT #1,N
80 CLOSE #1
```

Ecrire des données numériques (enregistrer l'ordre d'arrivée): résultat des courses

Pour chaque coureur que l'on connaît par son numéro de dossard (on lui demandera son nom plus tard, quand il sera moins essoufflé), il faut enregistrer l'ordre d'arrivée. En même temps, on connaîtra son nom et son prénom.
Dans chaque enregistrement dimensionné à quarante caractères, il reste dix caractères pour écrire des données supplémentaires.
Modifions donc l'enregistrement pour ajouter l'ordre d'arrivée de chaque coureur. Il faut (pour tous les enregistrements de 1 jusqu'à la fin du fichier):
— entrer le numéro d'enregistrement (dossard): N,
— lire l'enregistrement N,
— afficher le nom et le prénom,
— déposer l'ordre d'arrivée (ARRIV) dans l'enregistrement,
— réécrire l'enregistrement.

```
10 ' ARRIVEE1
20 OPEN "D",#1,"INSCRIPT",40
30 FIELD #1,15 AS NOM$,15 AS PRENOM$,4 AS ARRIV$
40 FOR ARRIV=1 TO LOF(1)
50   PRINT "ORDRE D'ARRIVEE:";ARRIV
55   INPUT "DOSSARD NUMERO:";N
60   GET#1,N
70   PRINT NOM$;PRENOM$
80   LSET ARRIV$=MKS$(ARRIV)
90   PUT #1,N
100 NEXT ARRIV
110 CLOSE #1
```

La ligne 80 vous étonne sans doute! L'instruction FIELD n'accepte que les variables chaînes de caractères. Il faut donc convertir la variable numérique ARRIV en chaîne de caractères. La première idée qui vient à l'esprit est STR$ (ARRIV). C'est un moyen. Mais le Basic comporte une fonction de conversion spéciale pour déposer un nombre dans une variable destinée à une instruction FIELD. Cette fonction, MKS$( ), donne une chaîne qui occupe toujours exactement quatre octets (si le nombre est en simple précision). Ainsi il est facile de préciser la taille à réserver pour inscrire l'ordre d'arrivée dans l'instruction FIELD: même si vous avez des dizaines de milliers de coureurs, il suffira de quatre octets pour inscrire l'ordre d'arrivée de la lanterne rouge.

Pour relire le fichier des coureurs, il faut convertir ARRIV$ en nombre: c'est CVS( ), la fonction inverse de MKS$, qui le réalise.

```
10 ' LECTURE APRES ARRIVEE
20 OPEN "D",#1,"INSCRIPT",40
30 FIELD #1,15 AS NOM$,15 AS PRENOM$,4 AS ARRIV$
35 DO
40   INPUT "NUMERO";N
50   IF N=0 THEN EXIT
60   GET #1,N
65   ARRIV=CVS(ARRIV$)
70   PRINT "NOM:";NOM$
80   PRINT "PRENOM:";PRENOM$
85   PRINT "ORDRE D'ARRIVEE:";ARRIV
90 LOOP
100 CLOSE #1
```

Si les nombres sont entiers (noter les temps)

Encore une amélioration, ensuite vous ferez les vôtres. Il reste un peu de place sur chaque enregistrement, profitons-en pour inscrire le temps en heures, minutes, secondes.
Il faut écrire trois nombres entiers. Le problème est le même que pour l'ordre d'arrivée: il faut convertir ces nombres en chaînes de caractères. Or il existe une fonction de conversion pour les nombres entiers: MKI$( ). Le résultat n'occupe que deux octets.
Plaçons donc les nombres d'heures, minutes, secondes dans des variables entières: H%, M%, S%.
Les chaînes de caractères correspondantes seront: MKI$(H%), MKI$(M%), MKI$(S%).

```
10 ' ARRIVEE2
20 OPEN "D",#1,"INCRIPT",40
30 FIELD#1,15 AS NOM$,15 AS PRENOM$,4 AS ARRIV$,2 AS HEURE$,2 AS MIN$,2
AS SEC$
40 FOR ARRIV=1 TO LOF(1)
50   PRINT "ORDRE D'ARRIVEE:";ARRIV
55   INPUT "DOSSARD NUMERO:";N
60   GET#1,N
```

```
70  PRINT NOM$;PRENOM$
80  LSET ARRIV$=MKS$(ARRIV)
82  INPUT"TEMPS(H,M,S)";H%,M%,S%
84  LSET HEURE$=MKI$(H%)
86  LSET MIN$=MKI$(M%)
88  LSET SEC$=MKI(S%)
90  PUT #1,N
100 NEXT ARRIV
110 CLOSE #1
```

Ainsi les quarante octets de chaque enregistrement sont occupés. Si vous vouliez ajouter d'autres informations, il faudrait recopier tous les enregistrements dans des enregistrements plus grands.
D'où l'intérêt de bien prévoir à l'avance ce qu'on veut enregistrer, ou de garder un peu de marge...

Pour relire les heures, minutes, secondes, nous allons utiliser la fonction CVI( ), inverse de MKI$( ).
Le programme de lecture après l'arrivée se modifie alors de la manière suivante:

```
30 FIELD #1,15 AS NOM$,15 AS PRENOM$,4 AS ARRIV$,2 AS HEURE$,2 AS MIN$,2
AS SEC$
68  H%=CVI(HEURE$):M%=CVI(MIN$):S%=CVI(SEC$)
88  PRINT"TEMPS:";H%;"H";M%;"M";S%;"S"
```

Voilà, nos améliorations sont terminées mais vous pouvez encore en faire vous-même. S'il vous venait à l'idée d'ajouter des nombres avec une très grande précision, voici encore quelques informations:
— les nombres en double précision occupent huit octets dans une variable de FIELD,
— on les convertit en chaîne de caractères par MKD$( ),
— la conversion inverse se fait par CVD( ).
Vous savez tout sur les fichiers à accès direct.

Utilisation de LSET et RSET

En principe, LSET et RSET sont destinés à déposer des données dans un enregistrement. Leur utilisation a été étendue, en dehors de la gestion des fichiers, au transfert de données d'une chaîne à une autre chaîne quelconque.

Ainsi est-il possible d'écrire:

```
LSET A$ = NOM$
```

même si la variable A$ n'a pas été définie dans un FIELD au préalable. Cette instruction transfère le contenu de NOM$ dans A$. Ceci suppose que A$ existe déjà et que sa longueur n'est pas nulle.

On peut par exemple la fixer par:

```
A$ = SPACE$(15)
```

Si la longueur de NOM$ est inférieure à celle de A$, le contenu est cadré à gauche dans A$, le reste étant comblé par des espaces. Si la longueur de NOM$ est supérieure à celle de A$, le contenu est tronqué à droite. Si les deux longueurs sont égales, tout va bien.
Il en va de même pour RSET, à ceci près que les données sont cadrées à droite.

Ceci peut être assez pratique pour transférer rapidement des données dans une zone de taille fixe. On peut également s'en servir pour cadrer des chaînes de caractères avant sortie sur imprimante, à la manière de PRINT USING.

En résumé:
— dans un fichier à accès direct tous les enregistrements ont la même longueur fixée une fois pour toutes;
— le transfert entre la mémoire centrale et la disquette se fait enregistrement par enregistrement en passant par une zone tampon;
— l'écriture dans la zone tampon peut se faire de deux manières:
• avec WRITE#, et il faut alors faire attention à ne pas "déborder" la zone tampon avec un enregistrement. On relit alors avec INPUT#.
• avec LSET (ou RSET): on dépose les données dans des variables de longueur déterminée par FIELD. Les variables numériques doivent alors être converties en chaînes de caractères (MKI$, MKD$, MKS$) et ces chaînes converties en nombres à la lecture (CVI, CVD, CVS).

Fonctions et instructions vues

PUT#	MKS$
GET#	CVS
LOF	MKI$
FIELD	CVI
LSET	MKD$
RSET	

Deuxième partie
Référence

Généralités

1. Mise en service

Pour utiliser l'interpréteur du Basic 128, il suffit de mettre le TO7-70 sous tension et de répondre 1 au menu affiché.
Le message

BASIC 128 v1.0 (c) Microsoft 1985
45695 bytes free
OK

apparaît ; ou 44591 avec un lecteur de disquette ; l'interpréteur Basic est actif.
L'interpréteur du Basic 128 est entièrement résident. Toutes les commandes, instructions et fonctions sont disponibles, y compris celles qui permettent de gérer des fichiers sur disquette.

La réponse 2 au menu affiché entraîne le chargement du programme nommé AUTO.BAT et son exécution, à partir de la disquette ou de la cassette.

2. Clavier et éditeur intégré

L'interpréteur Basic comporte un éditeur plein écran qui simplifie considérablement l'écriture et la modification de programmes.
Il est possible de modifier une ligne affichée à l'écran sans la retaper.
Après la modification, appuyer sur la touche ENTREE revient à introduire la nouvelle ligne dans le programme.

Clavier

Les touches spécifiques du clavier sont les suivantes:

STOP Arrête l'exécution du programme en cours. Rien n'est modifié sur l'écran. La reprise se fait en appuyant sur n'importe quelle touche sauf CNT , . et STOP.

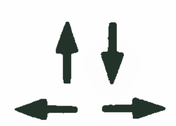
RAZ Efface l'écran et ramène le curseur en haut à gauche.

CNT S'utilise simultanément avec une autre touche:

CNT-C Interrompt l'exécution du programme ou de la commande en cours. Cette interruption n'intervient qu'à la fin de l'instruction en cours, ce qui peut durer quelque temps avec l'instruction PLAY.
Interrompt la numérotation automatique commandée par AUTO. La ligne courante n'est pas mémorisée.

CNT-X Efface les caractères, depuis la position du curseur jusqu'à la fin de la ligne.

CNT-W Place un lien logique entre la ligne où se trouve le curseur et la ligne qui suit. Cette facilité est utilisée pour rassembler deux lignes de programme en une seule.


Déplacement du curseur

Place le curseur en haut à gauche.

INS Bascule le mode d'insertion. Au passage en mode insertion, la caractère qui se trouve au-dessus du curseur passe en vidéo inverse. Les caractères tapés ensuite viennent se placer immédiatement à gauche du curseur. Un deuxième appui sur la touche INS ramène au mode normal.

EFF Supprime le caractère placé au-dessus du curseur.

Modification d'une ligne

Une ligne de programme ou une ligne de commandes ou d'instructions en mode direct peut comporter jusqu'à 255 caractères. Au fur et à mesure de la frappe, quand le curseur arrive au bord droit de l'écran, le retour au début de la ligne suivante se fait automatiquement.
Pour modifier une ligne présente entièrement à l'écran, il suffit de placer le curseur sur la partie à corriger et de taper les corrections soit en réécrivant sur les caractères à modifier, soit en supprimant, soit en insérant, soit les trois à la fois.
La nouvelle ligne est prise en compte en appuyant sur ENTREE, sans qu'il soit nécessaire d'amener le curseur en fin de ligne.

Création et modification d'un programme

Toute ligne commençant par un nombre compris entre 0 et 63999 est prise comme une ligne de programme. Son introduction dans le programme se fait quand on appuie sur ENTREE.
S'il existe déjà une ligne de même numéro, la nouvelle ligne la remplace.
Si la nouvelle ligne ne comporte que le numéro suivi ou non de caractères "espace", la ligne est supprimée.

Lors de l'entrée d'un programme, on peut obtenir une numérotation automatique des lignes par la commande AUTO.

La visualisation de tout ou partie d'un programme résidant en mémoire est faite avec la commande LIST.
La modification d'une ligne existante peut être faite soit en la réécrivant, soit en la modifiant directement si elle est visualisée.
La suppression d'une partie des lignes du programme se fait avec la commande DELETE.
La suppression de la totalité du programme est faite par la commande NEW.

3. Modes de fonctionnement

Quand l'interpréteur Basic est actif, le message "OK" apparaît. Dans cette situation, on peut utiliser Basic de deux manières différentes:

• Mode d'exécution directe

Ce mode permet d'effectuer des calculs, de donner des valeurs à des variables, d'appeler des fonctions ou d'exécuter un grand nombre d'instructions et toutes les commandes.

Chaque ligne est terminée par ENTREE et le résultat est obtenu immédiatement.
Ce mode peut servir à calculer comme avec une calculette évoluée, à tester certaines instructions ou à consulter certaines valeurs lors de l'arrêt d'un programme.

Exemples:

```
PRINT 12*12
144
OK
P=21.95
OK
PRINT P*1.1
24.145
OK
```

• mode programme

Pour entrer ou modifier un programme, il faut commencer chaque ligne par un numéro. Le numéro de ligne doit être compris entre 0 et 63999 inclus. Dans ce cas, la ligne est mémorisée et insérée dans le programme présent en mémoire.

L'exécution est obtenue par la commande RUN.

Une ligne de programme peut contenir des instructions, des commandes ou des commentaires. Cependant l'exécution de certaines commandes interrompt le déroulement du programme.

Exemple:

```
10 PRINT 31*24
RUN
744
OK
```

4. Structure d'une ligne

Une ligne de programme Basic doit avoir la forme générale suivante:

numéro instruction: instruction: instruction: ...

Le numéro de ligne est un nombre entier compris entre 0 et 63999.
La ligne de programme doit comprendre au moins une instruction.
Lorsqu'il y a plusieurs instructions, elles sont séparées par le caractère deux-points (:).
La longueur de la ligne ne doit pas dépasser 255 caractères. Il convient de tenir compte du fait que les minuscules accentuées sont codées sur trois caractères.
Les caractères qui suivent l'instruction REM ou le caractère apostrophe (') qui lui est équivalent, ne sont pas considérés comme des instructions et peuvent servir de commentaires. Des commentaires peuvent être ajoutés en fin de ligne après le caractère apostrophe (') ou après l'instruction REM.
Les espaces sont facultatifs sauf quand une variable précède un mot clé du langage. Il peut y avoir un nombre quelconque d'espaces entre un mot clé, une constante ou une variable.
Les mots clés et les variables peuvent être écrits indifféremment en majuscules ou en minuscules sans accents. L'interpréteur Basic transpose automatiquement les minuscules en majuscules.

exemple:

```
10 REM Je compte de 1 à 10
20 FOR I=1 to 10
30   PRINT I; ' affiche le nombre
40 next I
50 PRINT "c'est fini" : END
```

5. Alphabet utilisé

L'interpréteur Basic utilise le jeu de caractères ASCII comprenant 96 caractères affichables dont le code est compris entre 32 et 127. Les 32 premiers caractères peuvent être employés comme données en les introduisant directement au clavier (touche CNT+lettre) ou par la fonction CHR$.

Les minuscules accentuées et le ç sont utilisables à partir du clavier (frappe ACC+lettre ou ACC+signe+lettre) et sont codés sur trois caractères.

Les codes 128 à 255 sont destinés à recevoir les caractères définis par l'utilisateur au moyen de l'instruction DEFGR$, ils sont ensuite appelables par la fonction GR$(n) ou CHR$(128+n) avec n variant de 0 à 127.

6. Constantes

Il existe deux types de constantes:
— les constantes numériques,
— les constantes "chaînes de caractères".

Constantes numériques

Les constantes numériques sont des nombres positifs ou négatifs. Il existe trois types de constantes différents suivant la précision.

Entiers

Tous les nombres entiers positifs ou négatifs compris entre −32768 et +32767.
exemples:
13
−5273

Réels en simple précision

Tous les nombres positifs ou négatifs représentés sous la forme m*10e où la mantisse m a sept chiffres significatifs et où l'exposant e est compris entre −38 et +38. Dans la pratique, la précision exacte du nombre varie entre six et sept chiffres. On peut préciser,

mais ce n'est pas obligatoire, qu'une constante est en simple précision par le signe! à la fin.
exemples:
−1.6 E−19
6.02 E23
2.7828
12345!

Réels en double précision

Ils sont identiques aux réels en simple précision mais avec une mantisse de dix-sept chiffres significatifs. Dans la notation d'un nombre en double précision, la lettre marquant l'exposant est D au lieu de E. On peut, quand il n'y a pas d'exposant, préciser qu'une constante est en double précision par le signe # à la fin.
exemples:
3.1415926535#
.54321 D−27

Notation des constantes numériques

Les constantes numériques s'écrivent soit sous la forme décimale (12.3456), soit sous la forme "scientifique" (.123456E2).
Les nombres entiers peuvent être écrits en utilisant une base non décimale.

Notation hexadécimale
Une constante hexadécimale est un nombre exprimé en base 16. Il est précédé des caractères &H. Les chiffres sont 0,1,2,3,...,9,A,B,C,D,E,F.
exemples:
&HFFFF équivalent à 65535
&H32F

Notation octale
Une constante octale est un nombre exprimé en base 8. Il est précédé des caractères &O ou & seulement. Les chiffres sont 0,1,2,3,4,5,6,7.
exemples:
&O622
&1234

Notation binaire
Une constante binaire est un nombre exprimé en base 2, donc représenté uniquement de 0 et de 1. Il est précédé des caractères &B. Cette notation est particulièrement intéressante pour représenter les bits d'un octet ou d'une paire d'octets, spécialement lors de l'utilisation des opérateurs logiques (NOT, AND, OR, XOR, IMP, EQV).
exemples:
&B10101010
&B100000001

Constantes chaînes de caractères

Une constante chaîne de caractères est une suite de caractères encadrée par des guillemets ("). Le nombre de caractères de la chaîne, appelé longueur de la chaîne, doit être compris entre 0 et 255. Une chaîne qui ne contient aucun caractère est une chaîne vide.
Tous les caractères (du code 0 au code 255) peuvent être inclus dans une chaîne. Il faut remarquer que les minuscules accentuées et le ç quand ils sont introduits au clavier occupent trois caractères; il faut en tenir compte pour le calcul de la longueur.
exemples:
"QUE J'AIME A FAIRE CONNAITRE UN NOMBRE UTILE AUX SAGES"
"La concierge est dans l'escalier"

7. Variables

Les variables sont des emplacements mémoire qui possèdent un nom propre et qui sont destinés à recevoir les différentes données utiles au déroulement d'un programme. Une variable peut recevoir la valeur d'une constante ou le résultat d'un calcul ou d'une transformation de constantes et de variables.
Il y a quatre types de variables correspondant aux quatre types de constantes:
— variables numériques entières,
— variables numériques en simple précision,
— varaiables numériques en double précision,
— variables chaînes de caractères.

Nom des variables

Les noms de variables sont composés d'une lettre au moins, suivie éventuellement de lettres et de chiffres. Le nom d'une variable peut comporter jusqu'à 255 caractères. Toutefois, seuls les seize premiers sont significatifs et serviront à distinguer les variables entre elles. Deux variables dont les noms ont leurs seize premiers caractères identiques sont donc considérées comme identiques.
Le nom d'une variable ne peut pas commencer par un mot clé de Basic (ces mots sont appelés mots réservés, leur liste est donnée en Annexe 5). Ainsi TOTAL, FORCE, VALEUR ne peuvent pas être des noms de variable car ils commencent par TO, FOR, VAL.

exemples:
SOMME
A2
RECAPITULATIFDESDEPENSESDUMOIS
Le nom d'une variable doit être suivi de l'un des caractères (%, #, $) pour en préciser le type (entier,double précision ou chaîne de caractères). Le caractère ! peut être utilisé pour indiquer une variable en simple précision. Toute variable qui n'est pas terminée par %, #, $ est considérée comme étant en simple précision.
La signification de ces caractères est la suivante:

%	variable numérique entière
!	variable numérique en simple précision
#	variable numérique en double précision
$	variable chaîne de caractères

exemples:

QUOTIENT	variable simple précision
A3!	variable simple précision
PI#	variable double précision
NOBJET%	variable entière
ADR$	variable chaîne de caractères

Le type d'une ou de plusieurs variables peut être également précisé par l'une des instructions DEFINT, DEFSNG, DEFDBL, DEFSTR.

Tableaux

Le Basic offre la possibilité de regrouper un grand nombre de variables de même type sous le même nom dans un tableau.

Un tableau peut avoir une ou plusieurs dimensions. Chaque variable est accessible en donnant une valeur à chacun des indices et constitue un élément du tableau. On utilise quelquefois le terme variable indicée.
Tous les tableaux qui possèdent plus de 10 éléments doivent être déclarés au préalable par l'instruction DIM.
exemple:

```
DIM T(12)
```

Cette instruction crée un tableau numérique à une dimension de treize éléments. En effet, l'indice varie de la valeur 0 au maximum indiqué par DIM. T(3) désigne le quatrième élément du tableau T.

```
DIM PH$(2,10)
```

Cette instruction crée un tableau chaîne de caractères à deux dimensions. Le premier indice varie de 0 à 2, le second de 0 à 10. Il comporte donc 33 éléments.

La syntaxe du Basic accepte des tableaux ayant jusqu'à 255 indices, chaque indice pouvant aller de 0 à 32767.
Si l'évaluation d'un indice conduit à une valeur qui n'est pas entière, cette valeur est arrondie à l'entier le plus proche.
exemples:
A(3.5) est équivalent à A(4)
A(3.49) est équivalent à A(3)

Les noms des tableaux obéissent aux mêmes règles que les variables de même type:

%	entiers
!	simple précision
#	double précision
$	chaîne de caractères.

Volume occupé par une variable ou un tableau

Chaque type de variable correspond à une manière de représenter une valeur en mémoire. Le volume occupé en fonction du type de la variable est le suivant:

entier	2 octets
simple précision	4 octets
double précision	8 octets
chaîne de caractères	longueur de la chaîne en octets

Ainsi pour un tableau, il faut multiplier le volume occupé par un

élément par le nombre total d'éléments du tableau.
exemple:
DIM A#(19) réserve un tableau numérique en double précision dont les valeurs vont occuper 20*8=160 octets en mémoire.

La taille d'un tableau est limitée à 64 k-octets.

Conversions

Il est interdit de mélanger nombres et chaînes de caractères dans une expression.
Le Basic accepte des expressions numériques "mixtes", c'est-à-dire comprenant des variables et des constantes numériques de type différent.
L'évaluation d'une expression se fait alors en convertissant s'il y a lieu les valeurs dans le type le plus précis.
exemples:

```
? 12/23.5
.510638
```

L'entier 12 est converti en simple précision et la division est faite en simple précision.

```
? 2*3.141592653#
6.283185306
```

Après conversion de 2 en double précision, le calcul et son résultat sont en double précision.

Les opérations arithmétiques sont faites en double précision si l'un des opérandes comporte plus de sept chiffres significatifs ou s'il est noté en double précision.

Les fonctions mathématiques du langage (SIN, ATN, LOG,...) sont calculées en simple précision si leur argument est en simple précision et en double précision si leur argument est en double précision.
exemple:

```
?LOG(2.0001)
.693197
?LOG(2.0001#)
.693197179309987
```

Lors de l'affectation d'une valeur à une variable de précision différente, plusieurs cas sont à envisager:

— la valeur est de précision supérieure
Elle est convertie par arrondi (et non pas par troncature) à la précision de la variable.
exemple:

```
A%=251.72
?A%
252
```

— la valeur est de précision inférieure
Dans le cas d'un entier, il est converti en conservant sa valeur exacte. Par contre, un nombre en simple précision affecté à une variable en double précision est complété par des chiffres non significatifs mais non nuls.
exemple:

```
R#=.12345678E1
?R#
1.234567284584045
```

La valeur n'est améliorée en aucune mesure!
On peut remplir les derniers chiffres significatifs par des 0 en passant par la conversion des nombres en chaînes de caractères.
exemple:

```
R#=VAL(STR$(.12345678E1))
?R#
 1.234567
```

8. Expressions

Expressions numériques

Une expression numérique peut être une constante, une variable ou encore une combinaison de constantes, variables et fonctions reliées entre elles par des opérateurs.

Les opérateurs numériques sont classés en trois catégories:
— opérateurs arithmétiques,
— opérateurs de relation,
— opérateurs logiques.

Opérateurs arithmétiques

Voici la liste des opérateurs arithmétiques classés par ordre de priorité décroissante:

^	élévation à la puissance	X^Y
−	négation	−X
* /	multiplication, division	X*Y,X/Y
@	division entière	X@Y
MOD	modulo ou reste de la division entière	X MOD Y
+ −	addition, soustraction	

La priorité entre opérateurs signifie que dans une expression où deux opérateurs sont appliqués sans parenthésage, celui qui est prioritaire est évalué en premier. A priorité égale, l'évaluation commence par celui de gauche.
exemple:

```
? 4*3^2
36
? (4*3)^2
144
```

Division entière @

Cette opération n'est possible que si les opérandes sont compris entre −32768 et 32767. Avant la division, les deux opérandes sont arrondis au besoin. Le résultat est le quotient tronqué de la division.
exemples:

```
? 10@3
3
? 20@4.5
4
```

Cas d'erreurs

— Si un calcul amène au dépassement de la capacité dans le type considéré (valeur inférieure à −32768 ou supérieure à 32767 en entier, valeur inférieure à 10^−38 ou supérieure à 10^38 en réel) le message d'erreur Overflow apparaît.
— Une division par zéro est signalée par:
```
Division by zero
```
Dans les deux cas, l'exécution est interrompue.

Opérateurs de relation

Les opérateurs de relation sont utilisés pour comparer deux valeurs. Le résultat de la comparaison est une valeur “booléenne” VRAI ou FAUX. Par convention, VRAI a pour valeur −1 et FAUX a pour valeur 0.

Les opérateurs de relation sont les suivants:

=	égal à	X=Y
<>	différent de	X<>Y
<	inférieur à	X<Y
>	supérieur à	X>Y
<=	inférieur ou égal à	X<=Y
>=	supérieur ou égal à	X>=Y

Le résultat d’une relation peut être utilisé dans une instruction de branchement:

```
IF X+Y<A*B THEN GOSUB 200
```

Il importe de bien distinguer l’opérateur de relation = destiné à tester l’égalité de deux valeurs, de l’instruction = qui affecte une valeur à une variable.
exemple:

```
IF A=3 THEN B=2
```

signifie:
Si la valeur de A est 3 alors affecter 2 à B.

Puisque le résultat d’une comparaison est une valeur “booléenne” représentée par un nombre entier (0 ou −1), il peut être affecté à une variable numérique.
exemple:

```
RES = 2^3 < 3^2
?RES
 -1
IF RES THEN? "VRAI"
VRAI
```

Opérateurs logiques

Les opérateurs logiques permettent d'effectuer des opérations booléennes et par conséquent, de combiner plusieurs relations dans une condition.
Six opérateurs logiques sont disponibles:

NOT NON logique
AND ET logique
OR OU logique
XOR OU exclusif
IMP implication
EQV equivalence

NOT est le seul opérateur qui ne s'applique qu'à un seul opérande.
En prenant V pour VRAI et F pour FAUX, la table des opérateurs est la suivante:

X	Y	NOT X	X AND Y	X OR Y	X XOR Y	X IMP Y	X EQV Y
V	V	F	V	V	F	V	V
V	F	F	F	V	V	F	F
F	V	V	F	V	V	V	F
F	F	V	F	F	F	V	V

Noter que:
— l'expression X IMP Y est équivalente à (NOT X) OR Y
— l'expression X EQV Y est équivalente à NOT (X XOR Y)

Exemples :

```
IF A=B AND C=D THEN 500
IF NOT (A OR B) THEN 400
```

Fonctionnement des opérateurs logiques

En pratique, les opérateurs logiques travaillent sur des entiers (entre −32768 et 32767), c'est-à-dire sur des suites de seize bits.
Les opérations sont effectuées sur les bits de même position (un bit à 1 est VRAI, un bit à 0 est FAUX).
exemple:

```
? 127 AND 16
16
```

127 est représenté en binaire par 1111111
16 est représenté en binaire par 10000
le résultat vaut donc 10000

Cette particularité peut être utilisée pour extraire la valeur d'un bit dans un ou deux octets à la fois, à l'aide d'un masque.
Le masque &B11110000 ou encore 240 en décimal permet d'extraire avec AND les quatre bits de poids fort d'un octet.
Pour savoir si un nombre X% est pair, on peut faire:
X% AND &B1111111111111110 = X%
Le résultat vaut −1 (VRAI) si X% est pair,
0 (FAUX) si X% est impair.

Priorité des opérations

Lors de l'évaluation d'une expression numérique, l'ordre de priorité d'évaluation est le suivant:
1 - fonctions (ABS, SIN, INT,...)
2 - exponentiation ˆ
3 - négation −
4 - multiplication, division *,/
5 - division entière @
6 - modulo MOD
7 - somme, différence +, −
8 - relations >,>,=,<=,>=,<>
9 - opérations logiques NOT, AND, OR, XOR, EQV, IMP
On peut modifier l'ordre de priorité avec des parenthèses, les opérations situées dans les parenthèses les plus internes sont évaluées en premier.

Fonctions numériques

Les fonctions numériques disponibles (argument numérique, résultat numérique) sont les suivantes:

ABS(x)	valeur absolue de x
ATN(x)	arctangente de x
CDBL(x)	conversion en double précision
CINT(x)	conversion en entier
COS(x)	cosinus de x
CSNG(x)	conversion en simple précision
EXP(x)	exponentielle de x
FIX(x)	partie entière de x
INT(x)	plus grand entier inférieur ou égal à x

LOG(x)	logarithme népérien de x
MAX(x,y,...)	maximum de x,y,...
MIN(x,y,...)	minimum de x,y,...
RND(x)	générateur de nombres aléatoires
SGN(x)	signe de x
SIN(x)	sinus de x
SQR(x)	racine carrée de x
TAN(x)	tangente de x

Expressions chaînes de caractères

Une expression chaîne de caractères peut être une constante, une variable ou encore une combinaison de constantes, variables et fonctions reliées entre elles par des opérateurs.
Il n'existe qu'un seul opérateur sur chaîne de caractères:
\+ concaténation
Cet opérateur permet de mettre deux chaînes bout à bout pour en faire une troisième.
exemples:

```
? "AUTO" + "ROUTE"
AUTOROUTE
A$="ETRE ": B$="OU ": C$="NE PAS "
? A$+B$+C$+A$
ETRE OU NE PAS ETRE
```

La concaténation ne place pas d'espace entre les deux chaînes. Il faut donc mettre des espaces à la fin de chaque mot pour faire une phrase.

Comparaison de chaînes de caractères

Les opérateurs de relation peuvent servir à comparer des chaînes de caractères, mais le résultat de la comparaison est logique, donc exprimé sous forme numérique: −1 (VRAI) ou 0 (FAUX).
La comparaison se fait caractère par caractère, à partir du début de chaque chaîne. Un caractère est considéré comme plus petit qu'un autre si son code dans le jeu de caractères ASCII est inférieur.
Tant que les caractères sont égaux, la comparaison se poursuit. La chaîne qui possède le premier caractère inférieur au caractère correspondant dans l'autre chaîne est considérée comme la plus petite.

Si la fin de l'une des deux chaînes est atteinte avant qu'une différence soit apparue, la chaîne la plus courte est considérée comme plus petite.
Dans une comparaison, tous les caractères sont pris en compte, y compris les espaces et les caractères non visualisables.
exemples:
"AB" < "FG" car A est inférieur à F
"ABCD" < "ABCDEF" car "ABCD" est plus courte
"GRAND" < "GROS" car A est inférieur à O
LOUIS" < "Louis" car O est inférieur à o

Les expressions chaînes peuvent aussi comporter des fonctions dont le résultat est une chaîne (LEFT$, MID$,...).

9. Entrées-sorties et fichiers

Entrées-sorties généralisées

La gestion des échanges avec les principaux périphériques (cassettes, disquettes, clavier, écran, imprimante et communication série) est faite par le même jeu d'instructions à usage général. Ces instructions permettent de gérer les transferts concernant programmes, données sous forme ASCII et données sous forme binaire.
L'orientation des informations vers un périphérique ou un autre est faite par le choix du nom de périphérique, de ses options de fonctionnement et du nom de fichier à transférer. Cet ensemble de noms constitue le descripteur de fichier.

Descripteur de fichier

Le descripteur de fichier est une chaîne de caractères qui comprend trois parties organisées de la manière suivante:

Nom du périphérique: (options) nom du fichier

Suivant les cas, chacune de ces trois parties peut être facultative.

Nom du périphérique et options

Les noms de périphériques sont les suivants:

0	1er lecteur de disquettes
1	2e lecteur de disquettes
2	3e lecteur de disquettes
3	4e lecteur de disquettes
CASS	magnétophone à cassettes
COMM	voie de communication série
KYBD	clavier
LPRT	imprimante parallèle
SCRN	écran

Le nom du périphérique, quand il est présent, doit toujours être suivi de deux points (:).

Si le nom du périphérique n'est pas précisé, sa valeur par défaut est:

SCRN: avec la commande LIST et TRON

0: si une unité de disquettes est connectée, pour toutes les autres commandes et instructions

CASS: si seul le lecteur-enregistreur de cassettes est connecté

L'instruction DEVICE permet de modifier le périphérique pris par défaut dans une instruction d'entrée ou de sortie.

Options

Trois périphériques acceptent des options: l'imprimante parallèle (LPRT:), la voie de communication série (COMM:) et les unités de disquettes (0: 1: 2: 3:).

Imprimante parallèle:

L'option indique le nombre de caractères par ligne: entre 0 et 255. Par défaut, ce nombre est fixé à 40. 0 indique une ligne infinie exemple:

LPRT:(80) désigne une imprimante à 80 colonnes.

Voie de communication série:

L'option comporte obligatoirement trois parties.

— le 1er chiffre indique la vitesse de transmission

1	110 bauds
2	300 bauds
3	600 bauds
4	1 200 bauds
5	2 400 bauds
6	4 800 bauds

— le 2e chiffre indique le nombre de bits transmis
7 transmission sur 7 bits
8 transmission sur 8 bits
la parité n'est pas gérée lors de la transmission série; elle doit donc être comptée dans le nombre de bits transmis.
— le 3e chiffre et les suivants indiquent le nombre de caractères par ligne (entre 0 et 255) comme pour l'imprimante parallèle, 0 correspondant à une ligne infinie. Un code de retour à la ligne est émis au bout du nombre de caractères indiqué.
exemple:
COMM:(47120) précise une transmission à 1 200 bauds (4) sur 7 bits (7) avec 120 caractères par ligne (120).

Unité de disquettes

L'option est un commentaire associé au fichier lors de son écriture sur la disquette. Ce commentaire est une chaîne de huit caractères maximum.
Ce commentaire est affiché sur la droite de l'écran quand on demande le catalogue de la disquette (DIR).
exemple:

```
SAVE "(13 mai)TEST"
```

le commentaire 13 mai est associé au fichier TEST.BAS .

Nom du fichier

Le nom d'un fichier est composé de deux parties séparées par un point (.):
nom . suffixe
Le nom comporte huit caractères au plus: tous les caractères sont permis à l'exclusion du code 0, du code 255, de deux parenthèses et du point.
Le suffixe comporte trois caractères au plus avec les mêmes restrictions que le nom. Le suffixe n'est pas obligatoire.
exemples:
ECHECS.JEU

Euclide.tst
ESSAI

En l'absence de suffixe, l'interpréteur Basic ajoute le suffixe suivant:

.BAS	pour un programme Basic
.DAT	pour un fichier de données
.BIN	pour un fichier binaire (zone mémoire ou programme en langage machine)
.MAP	pour une image de l'écran
.TRA	pour une trace d'exécution de programme

Il existe un seul nom de fichier ayant un usage particulier:
AUTO.BAT
Ce nom est réservé à un programme Basic qui s'exécute automatiquement au chargement du Basic (choix D du menu initial).

Autres périphériques

Le son

La génération de sons est programmable en Basic sur une plage de cinq octaves.
Les sons transmis au haut-parleur du téléviseur sont les notes de la gamme (avec dièses et bémols) et le silence.
Pour chaque note émise, il est possible de régler: la durée (de la ronde à la double croche pointée), le tempo (sur une plage de 256 valeurs) et l'attaque qui rend le son plus ou moins amorti.
Les sons sont définis dans des chaînes de caractères. L'ordre d'émission sur le haut-parleur est donné par l'instruction PLAY.

Le crayon optique

Le crayon optique comprend un capteur qui analyse le rayonnement émis par le balayage du tube et un interrupteur qui sert à valider l'instant de la mesure.

La mesure du temps qui s'écoule entre le début du balayage et la perception du rayonnement permet de déterminer l'endroit visé avec une très bonne précision, égale au point élémentaire de l'image.

Le relevé des coordonnées du point visé par le crayon optique se fait par deux instructions:

INPEN relevé à distance

INPUTPEN relevé lors du contact de l'interrupteur

La position de l'interrupteur peut être connue par la fonction PTRIG.
Le contrôle d'une séquence d'instructions par la visée d'une zone au crayon optique est possible en définissant les zones par l'instruction PEN et en faisant les branchements correspondants par ONPEN...GOTO ou ONPEN...GOSUB.
Le noir et le rouge ne permettent pas de viser correctement un point.
En règle générale, la luminosité doit atteindre une valeur minimale pour que le crayon optique puisse fonctionner.
Pour obtenir une mesure précise, il est nécessaire de régler le crayon optique à la mise en service.

Les manettes de jeu

On peut connecter deux manettes de jeu sur le contrôleur de jeu.
La position de chacune de ces deux manettes est donnée par deux fonctions:

STICK position de la manette dans l'une des 8 directions

STRIG état du bouton "action" de la manette

10. Visualisation: mode caractère et mode graphique

La visualisation des données sur l'écran peut se faire sous deux formes différentes:

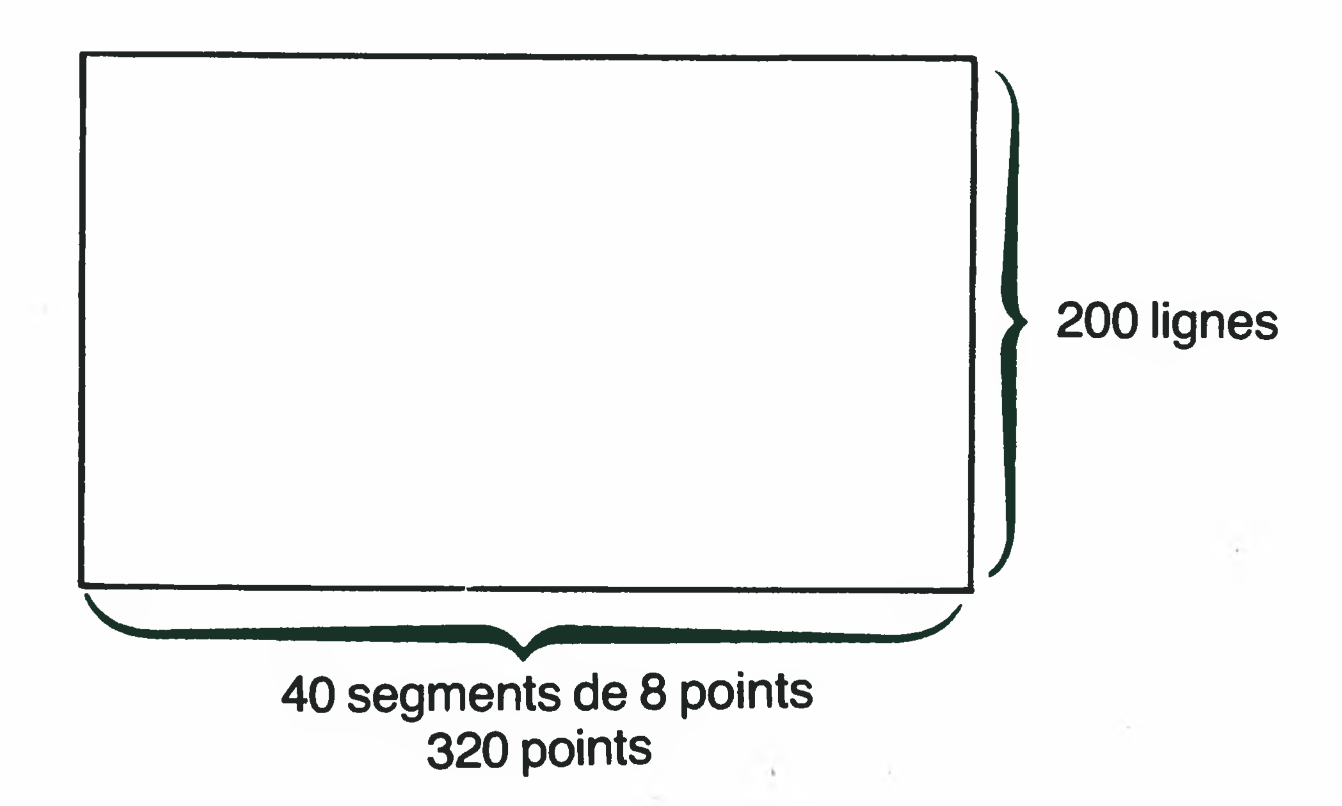
— *mode caractère:* pour afficher du texte ou des symboles prédéfinis de la taille d'un caractère,

— *mode graphique:* pour afficher des traits, des points, des boîtes ou des objets quelconques.

Les deux modes fonctionnent simultanément, ce qui permet d'afficher sur le même écran, du texte et du graphique.

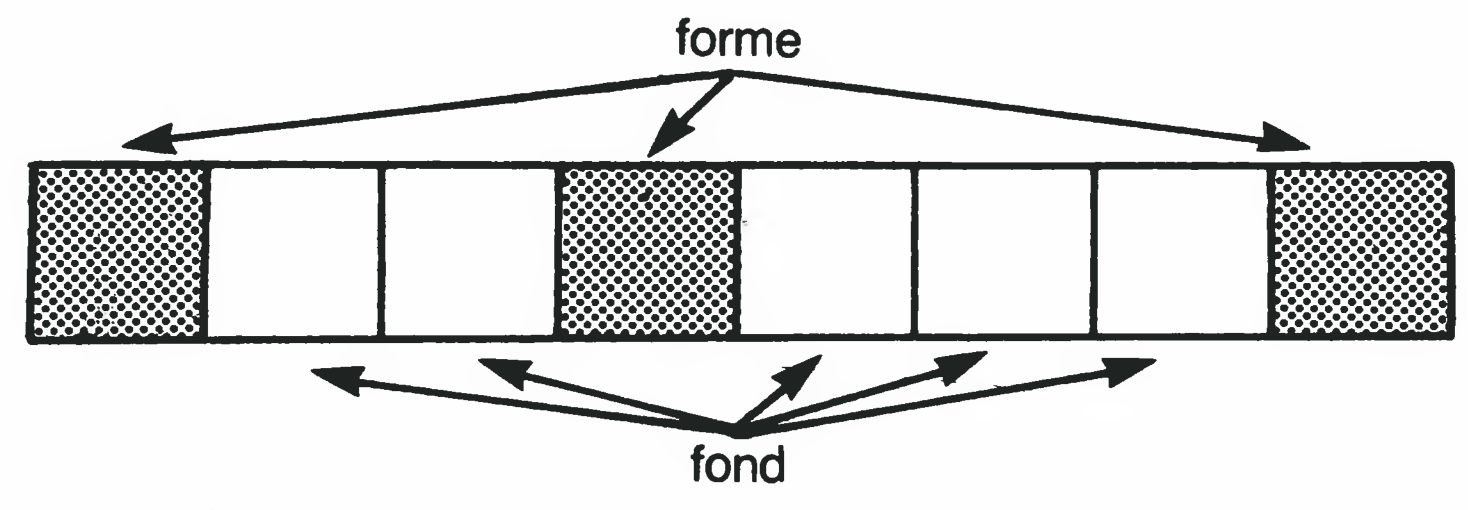
Fonctionnement de la visualisation

La partie utile de l'écran est composée de 200 lignes distinctes. Chaque ligne est divisée en 40 segments de 8 points chacun.



A l'intérieur d'un segment, chaque point est rattaché à l'une des deux catégories:
— forme: il s'agit d'un point appartenant à un caractère, à une ligne, à une boîte ou à un objet.
— fond: c'est un point du fond de l'écran, qui n'a été modifié par aucun tracé ou aucun affichage ou qui a été remis au "fond".

exemple d'un segment avec 3 points de forme



Pour un même segment, on ne peut attribuer que deux couleurs:
— une couleur pour les points de la forme,
— une couleur pour les points du fond.
Les deux couleurs d'un segment (forme et fond) sont celles qui ont été fixées par la dernière programmation d'un point de forme ou de fond du segment.
Si aucun point du segment n'a été modifié par une instruction d'affichage ou de tracé, le segment entier appartient au fond de l'écran.

L'appartenance d'un point à la forme ou au fond lors d'un tracé est fixée par le code employé pour la couleur:
— code positif: le point appartient à la forme,
— code négatif: le point appartient au fond.
Si la programmation d'un point de l'écran est faite avec une instruction dans laquelle la couleur n'est pas précisée, le point est traité comme un point de la forme et la couleur prise par défaut est celle qui a été choisie précédemment pour les caractères (instructions SCREEN et COLOR).
D'un segment de huit points à l'autre, le choix des deux couleurs est entièrement libre. Les segments sont indépendants.
Il est possible d'afficher et de tracer sans modifier la couleur de forme des segments atteints par le tracé (troisième paramètre de l'instruction CONSOLE).
L'inversion des couleurs de forme et de fond simultanément sur tout l'écran est possible (quatrième paramètre de l'instruction SCREEN).
L'inversion de la suite de l'affichage est possible également (3e paramètre de l'instruction COLOR).

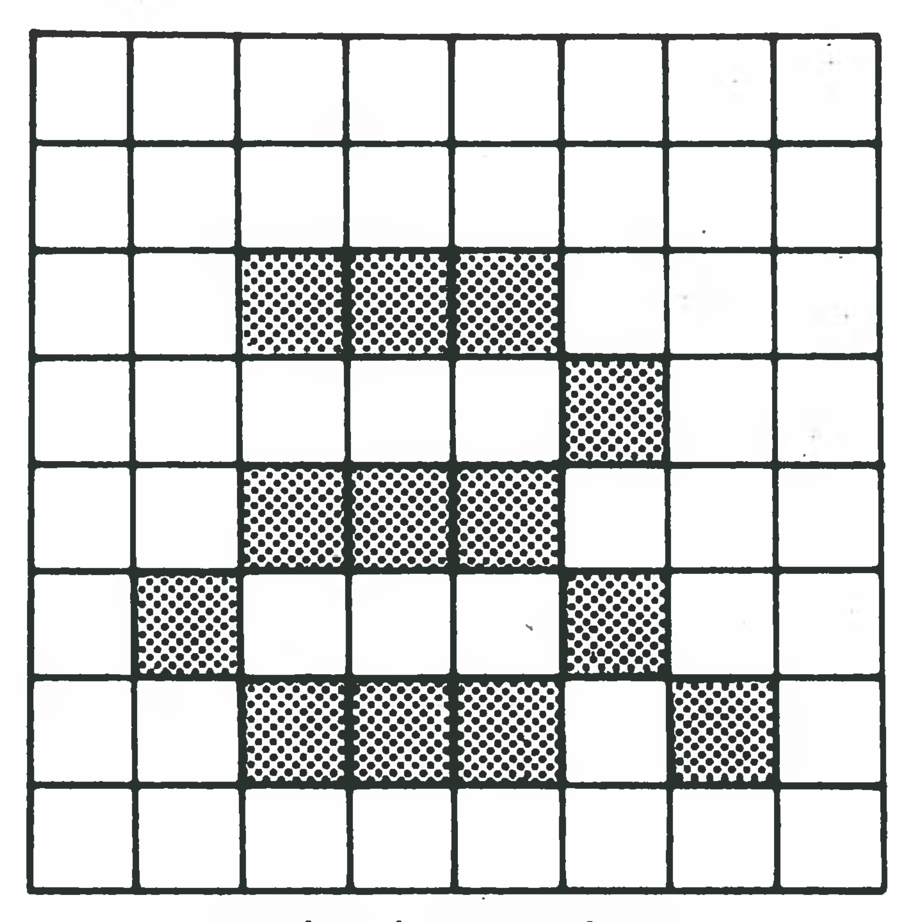
Choix des couleurs

couleur	code forme	code fond
noir	0	−1
rouge	1	−2
vert	2	−3
jaune	3	−4
bleu	4	−5
magenta	5	−6
cyan	6	−7
blanc	7	−8
gris	8	−9
rouge clair	9	−10
vert clair	10	−11
jaune clair	11	−12
bleu clair	12	−13
magenta clair	13	−14
cyan clair	14	−15
orange	15	−16

Les couleurs claires (de 8 à 14) sont les mêmes que les sept premières mais avec un peu de blanc.
Le code de fond d'une couleur s'obtient facilement par la relation:
code fond = −(code forme + 1)

Mode caractère

Dans les instructions fonctionnant sur le mode caractère, l'écran comporte 25 lignes de 40 (ou 80)caractères chacune. Chaque caractère lui-même est représenté sur l'écran par une matrice de 8*8 points, soit encore huit segments.



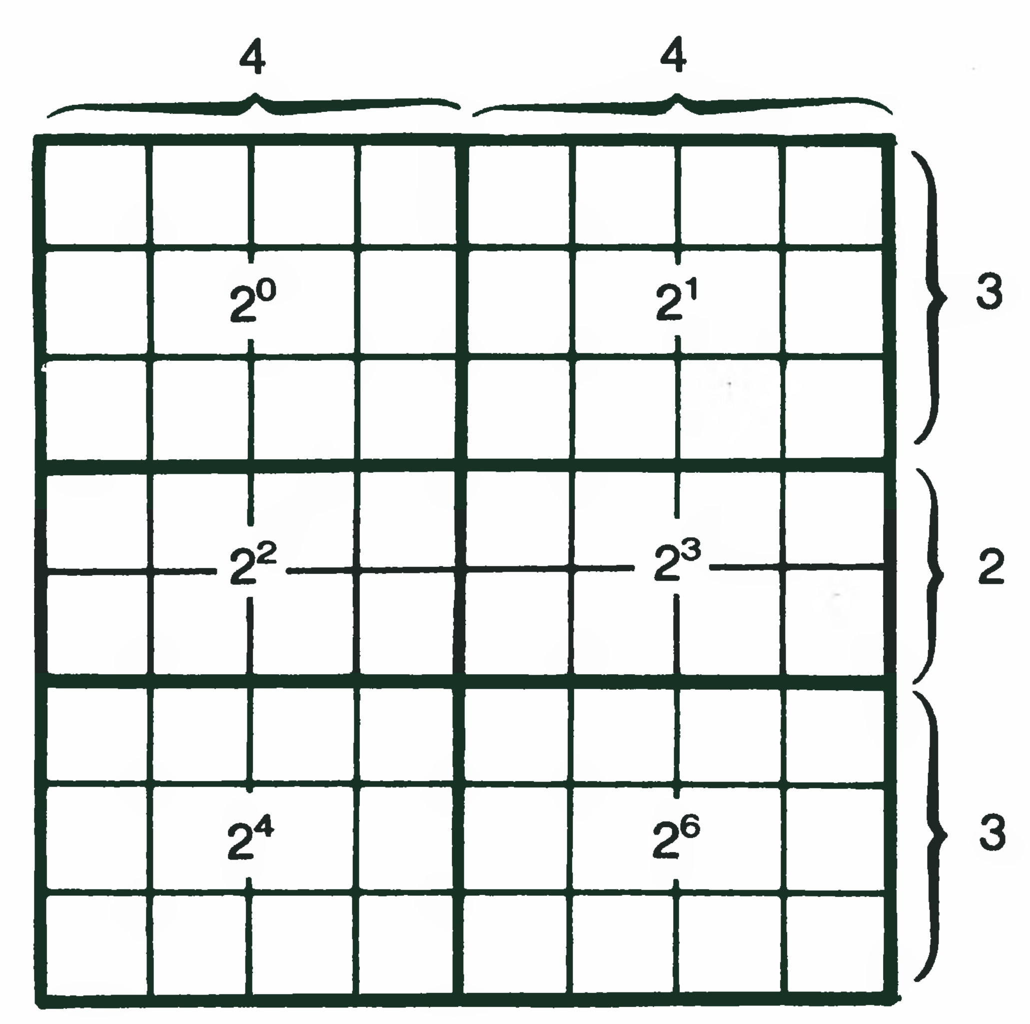
matrice du caractère a

En mode caractère, les lignes sont comptées de 0 (ligne supérieure) à 24 (ligne inférieure de l'écran) et les colonnes sont comptées de 0 (la plus à gauche) à 39 (la plus à droite).
L'instruction LOCATE permet de placer une chaîne de caractères en n'importe quel point de l'écran.
La position du curseur qui marque l'endroit où doit se faire le prochain affichage, est donnée par deux fonctions: CSRLIN pour le numéro de la ligne et POS pour le numéro de la colonne.
Le code ASCII du caractère situé en une position quelconque de l'écran est donné par la fonction SCREEN (à ne pas confondre avec l'instruction du même nom).
La fenêtre d'affichage peut être limitée en hauteur entre une ligne supérieure et une ligne inférieure par l'instruction CONSOLE (premier et deuxième paramètres).
Les caractères peuvent être affichés en double hauteur et double largeur au moyen de l'instruction ATTRB.
Certaines instructions graphiques fonctionnent en mode caractère, c'est-à-dire qu'elles effectuent le même type de tracé mais en prenant un caractère à la place d'un point et avec les coordonnées du mode caractère (colonnes de 0 à 39 et lignes de 0 à 24).:

PSET, LINE, BOX, BOXF. Les caractères peuvent être utilisés pour remplir une zone en mode graphique. Le caractère est alors pris comme motif de remplissage (instruction PATTERN).
Au cours du déroulement d'un programme, plusieurs chaînes de caractères peuvent être "masquées" (troisième paramètre de l'instruction ATTRB). Les emplacements des caractères masqués sont alors remplis de noir. Les caractères sont démasqués par l'instruction UNMASK; ils prennent alors la couleur en cours au moment du masquage.

Caractères semi-graphiques vidéotexte

Le jeu de caractères semi-graphiques vidéotexte est disponible en Basic.
Chaque caractère de ce jeu est divisé en six zones. Le poids de chaque zone est fixé par la position d'un bit dans le code du caractère.



découpage du caractère vidéotexte

Le code du caractère est obtenu en mettant le bit à 1 si la zone correspondante est "allumée" c'est-à-dire appartient à la forme, et à 0 si la zone est éteinte, ou appartient au fond. Par convention le bit 7 est à 0 et le bit 5 est à 1.

0		1					
b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0

Codage d'un caractère vidéotexte

Les caractères semi-graphiques ne sont affichables qu'en taille normale; ils ne sont pas affectés par l'instruction ATTRB.
L'affichage de ces caractères se fait entre deux caractères de contrôle par l'instruction PRINT:
CHR$(14) passage en mode semi-graphique
CHR$(15) retour au mode normal

Le retour au mode normal se fait automatiquement à la fin d'un programme.

Exemple:

```
? CHR$(14);CHR$(102);CHR$(57);CHR$(15)
```

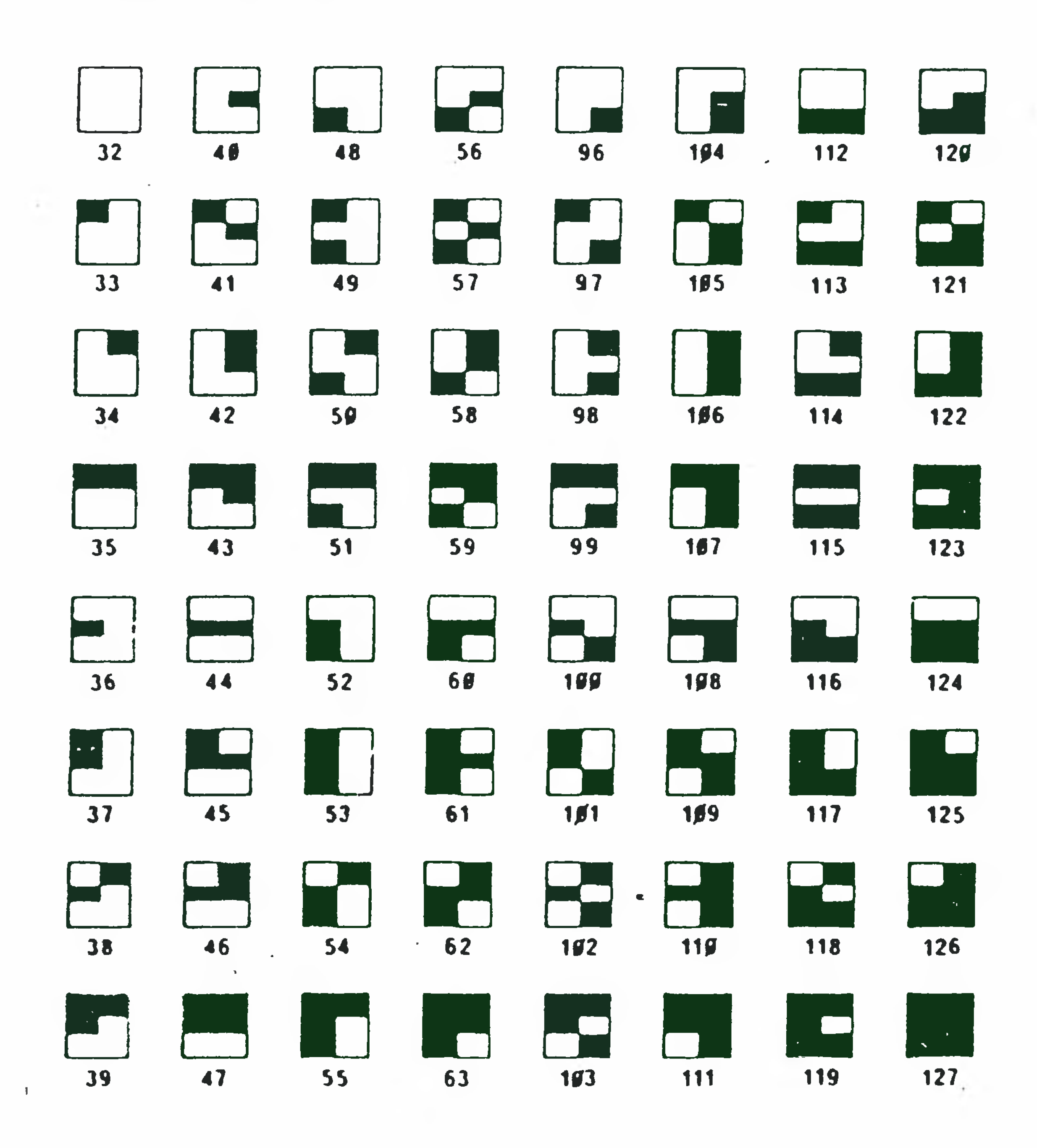
ou encore:

```
? CHR$(14);f9;CHR$(15)
```

Chacun des codes semi-graphiques peut être remplacé par la lettre équivalente du tableau ASCII.
102 ------- f
57 ------- 9

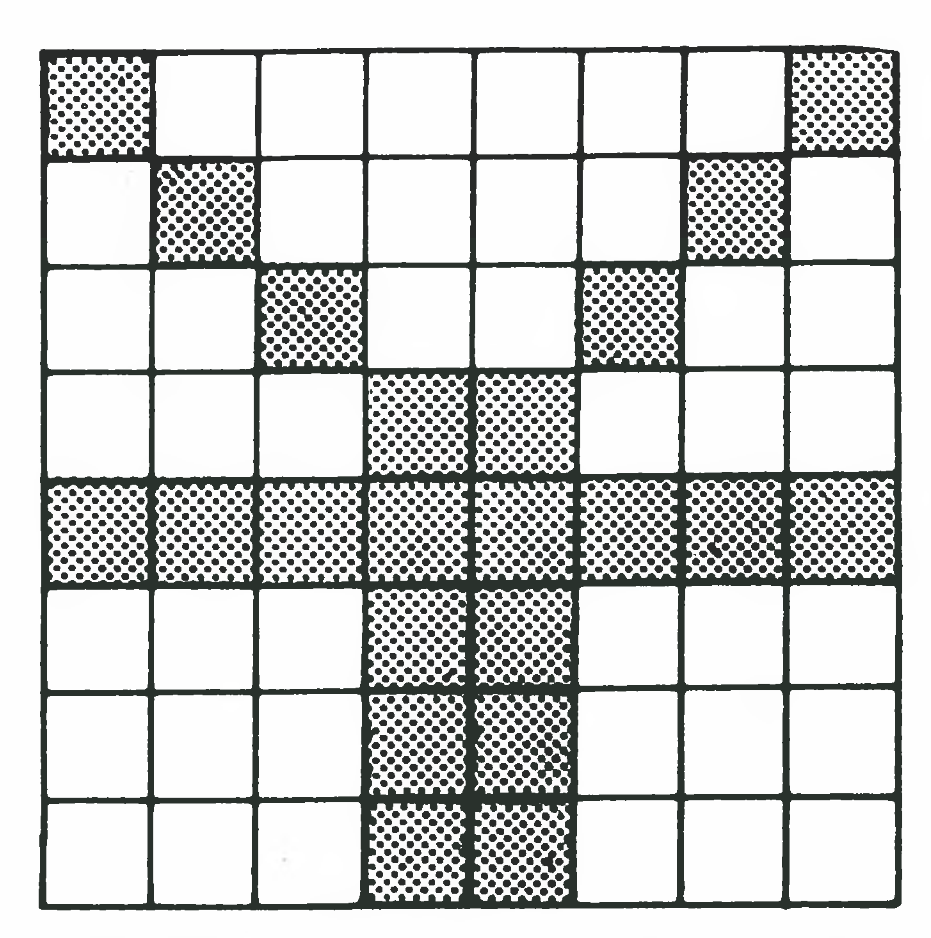
CARACTERES SEMI-GRAPHIQUES TELETEL

Code
décimal



Caractères définis par l'utilisateur

Il est possible de définir 128 caractères de forme quelconque et qui pourront ensuite être utilisés comme des caractères normaux.
Ces caractères sont définis par l'utilisateur au moyen de l'instruction DEFGR$. Le caractère numéro i est ensuite désigné par GR$(i) ou CHR$(128+i). Les numéros de caractères utilisateur s'étendent de 0 à 127.
Ils sont affichés par l'instruction PRINT ou PRINT USING. Ils peuvent être doublés en hauteur et en largeur par ATTRB.
Le nombre maximum de caractères à définir est fixé, le plus souvent en début de programme, dans une instruction CLEAR (troisième paramètre).
Si le nombre de caractères réservés est n, le numéro des caractères s'étend de 0 à n−1.
La définition d'un caractère est faite par huit nombres entiers compris entre 0 et 255. Chaque nombre entier représente l'un des huit segments de huit points qui composent le caractère.
Pour chaque segment, la valeur associée est calculée en binaire en prenant 1 si le point est allumé c'est-à-dire appartient au dessin du caractère, et en prenant 0 si le point appartient au fond.



matrice de points du caractère numéro i

binaire	décimal
1000 0001	129
0100 0010	66
0010 0100	36
0001 1000	24
1111 1111	255
0001 1000	24
0001 1000	24
0001 1000	24

valeur des segments du caratère i

Chaque caractère peut être redéfini en cours de programme.
exemple:

```
10 ' Définition d'un caractère
20 CLEAR ,,1
30 DEFGR$(0) = 129,66,36,24,255,24,24,24
40 PRINT GR$(0)
50 PRINT
60 ATTRB 1,1 : PRINT CHR$(128)
```

Mode graphique

En mode graphique, l'écran est composé de 200 lignes de 320 points chacune.
Les coordonnées d'un point graphique visible sont comprise:
— pour les colonnes entre 0 et 319
— pour les lignes entre 0 et 199
Les coordonées d'un point sont toujours indiquées avec la syntaxe suivante:
(colonne , ligne)
Le point (0,0) est situé en haut à gauche et le point (319,199) en bas à droite.
Toutes les instructions graphiques acceptent des coordonnées en dehors de la zone visible. Les valeurs des coordonnées d'un point en colonne et en ligne peuvent varier de −32768 à 32767.
La partie visible des tracés graphiques peut être réduite à une taille inférieure à celle de l'écran; la fenêtre visible est définie par l'instruction WINDOW.
Si la valeur d'une coordonnée n'est pas entière, elle est arrondie à l'entier le plus proche.

Les instructions de tracé sont les suivantes

PSET	point
LINE	segment
BOX	boîte
BOXF	boîte pleine
CIRCLE	cercle ou ellipse
CIRCLEF	cercle ou ellipse pleins

La couleur du tracé est la dernière fixée par SCREEN ou COLOR ou bien elle peut être précisée dans l'instruction.
Si le code de couleur est négatif, les points tracés appartiennent alors au fond.
Le tracé peut être effectué de trois manières différentes:
— avec effacement: le tracé efface ce qui se trouvait précédemment au même endroit,
— transparent: le tracé se superpose au tracé précédent, les points de forme du tracé précédent ne sont pas effacés par les points de fond du nouveau tracé,
— avec inversion: chaque point du tracé est inversé : s'il appartenait au fond, il passe à la forme et vice versa. Deux tracés identiques au même endroit rétablissent l'état initial.
De plus, le tracé peut être fait avec ou sans modification de la couleur des points.
Toutes ces modalités sont fixées par le troisième paramètre de CONSOLE.

La fonction POINT permet de connaître la couleur d'un point et son état: le code retourné est positif si le point appartient à la forme et négatif s'il appartient au fond.

Le remplissage d'une surface de couleur homogène avec une couleur quelconque est possible avec l'instruction PAINT. Par défaut, le remplissage est uniforme. Le motif de remplissage (caractère ASCII ou caractère défini par l'utilisateur) est fixé par l'instruction PATTERN.

Plusieurs zones de l'écran peuvent être mémorisées dans un tableau numérique entier puis sauvegardées dans un fichier pour être utilisées ultérieurement (instructions GET, SAVEP, LOADP et PUT). Les coordonnées de chaque zone à sauvegarder doivent correspondre au découpage de l'écran en caractères (colonnes et lignes comptées en caractères).

11. Tortues

On peut définir jusqu'à dix tortues (numérotées de 0 à 9) dont le comportement est semblable à celui de la tortue Logo.
Chaque tortue est caractérisée par:
— sa forme,
— sa taille,
— sa position,
— sa direction de mouvement,
— son orientation,
— sa trace (ou non),
— sa présence (visible ou invisible).
Chaque tortue peut être déplacée en translation dans sa direction, en rotation autour de son centre; sa direction de mouvement peut être modifiée.

La forme d'une tortue est définie par des segments visibles ou invisibles avec l'instruction TURTLE.
La taille d'une tortue est modifiée par l'instruction ZOOM, soit en absolu, soit en relatif.
La position d'une tortue, en dehors de tout déplacement, est fixée en coordonnées absolues par l'instruction TURTLE.

Le déplacement d'une tortue dans sa direction est commandé par l'instruction FWD, en avant comme en arrière.
La direction du déplacement est fixée ou modifiée de façon absolue ou relative par l'instruction HEAD.
La tortue peut être placée et déplacée dans un écran virtuel dont les coordonnées varient de −32768 à 32767.
Cet écran s'enroule sur lui-même, le dépassement des bornes d'un côté ramenant la tortue du côté opposé.
L'orientation de la tortue elle-même est fixée ou modifiée par l'instruction ROT, de façon absolue ou relative.
L'instruction TRACE indique si la tortue laisse une trace de ses déplacements.
L'instruction SHOW indique d'une part si la tortue est visible ou invisible et d'autre part si les modifications de taille et d'orientation sont à effet immédiat ou différées au prochain déplacement (FWD) ou placement (TURTLE).
Les valeurs des angles dans les instructions HEAD et ROT sont données en 1/256e de cercle. La valeur 0 correspond à 0°, 128 à 180° et 255 à (360*255)/256 degrés.

Par défaut, les caractéristiques d'une tortue sont les suivantes:

forme	triangle isocèle
taille	echelle normale (16)
position	au centre de l'écran
direction	vers la droite de l'écran (0°)
orientation	identique à la direction (0)
trace	crayon levé
visibilité	invisible
évolution	modifications différées

L'état d'une tortue donnée peut être entièrement connu au moyen d'une instruction:

INPUTTURTLE	position

et de cinq fonctions:

HEAD	direction
ROT	orientation
ZOOM	taille
SHOW	visibilité
TRACE	trace

Les instructions et fonctions s'appliquent à la tortue active; celle-ci est déterminée par la dernière instruction TURTLE.

12. Gestion des erreurs

Les erreurs sont détectées par l'interpréteur Basic au moment de l'exécution des instructions. La détection d'une erreur provoque l'arrêt de l'exécution.
Chaque erreur donne lieu à un court message à l'écran.
exemples:

```
Syntax Error
Division By Zero
Missing Operand
```

Si l'erreur apparaît dans un programme, le numéro de la ligne en cours d'exécution est indiqué.
exemple:

```
Type Mismatch in 150
```

L'affichage de la ligne où l'erreur s'est produite peut être obtenu par:

```
LIST .
```

Un pavé en vidéo inverse est inséré automatiquement après l'endroit de la ligne où l'erreur a été détectée.

Pour éviter que certaines erreurs prévisibles ne viennent perturber le fonctionnement d'un programme et pour éventuellement y remédier, une instruction ON ERROR ... GOTO permet de déclarer le début d'une séquence de traitement d'erreur.
Dans la séquence de traitement d'erreur, le numéro de l'erreur est contenu dans la variable ERR et le numéro de la ligne où elle s'est produite dans la variable ERL.
La liste des messages d'erreur et de leurs numéros est donnée en Annexe 7.
Après traitement de l'erreur, l'instruction RESUME permet de reprendre l'exécution de la suite du programme soit à l'endroit où l'erreur s'est produite, soit à un endroit différent.

exemple:

```
10 ' DIVISION PAR 0
20 ON ERROR GOTO 100
30 DO
40   INPUT X
50   PRINT 1/X
60 LOOP
90 END
100 IF ERR=11 THEN PRINT "Donner un nombre non"
110 RESUME 40

RUN
? 4
2.5
? 0
Donner un nombre non nul
```

Liste aplhabétique des commandes

Instructions et fonctions

Avant d'entrer dans le détail de chaque commande, instruction ou fonction, rappelons quelques conventions de notations.

— Les éléments facultatifs d'une instruction sont imprimés sur le fond coloré.
— Les principaux paramètres qui changent d'une utilisation à l'autre sont désignés par:

Nombre: représente une constante numérique (3*55) ou une variable numérique (Z) ou une expression dont le résultat est numérique (Z*RND)

Chaîne de caractères: représente une constante ("JULES") ou une variable (P$) ou une expression dont le résultat est une chaîne de caractères (MID$("JULES", 1,1)).

Donnée: représente un nombre ou une chaîne de caractères.

Variable: représente une variable numérique (X) ou une variable chaîne de caractères (Y$).

— Les différents exemples fournis avec chaque instruction illustrent la partie optionnelle de l'instruction. En général, les options sont précisées dans l'explication.

ABS(nombre)
fonction

Donne la valeur absolue du nombre.

Exemple:

```
? ABS(5 * 3)
15
```

ASC(chaîne de caractères)
fonction

Donne le code ASCII du premier caractère de la chaîne.

Si la chaîne de caractères est vide, le message d'erreur Illegal Function Call apparaît.

Remarque: les minuscules accentuées et le ç sont codés par une séquence de trois caractères dont le premier a toujours le code 22. Ainsi, le caractère é est codé par la séquence de codes 22, 66, 101. On obtient chacun de ces codes en utilisant la fonction MID$.

Exemple:

```
?ASC("COLEOPTERE")
67
?ASC(MID$("é",2,1))
66
```

ATN(nombre)
fonction

Donne l'arctangente du nombre.

Le résultat de la fonction est un angle, exprimé en radians, compris entre $-\pi/2$ et $\pi/2$.
Si le nombre est en double précision, le résultat est donné en double précision.

Exemple:

```
?ATN(3)
1.24905
```

ATTRB largeur, hauteur, masque
instruction

Définit la taille des caractères affichés sur l'écran et fixe le mode d'affichage: normal ou masqué.

Le 1er paramètre indique la largeur des caractères:
largeur = 0, largeur normale
largeur = 1, largeur double
Le 2e paramètre indique la hauteur des caractères:
hauteur = 0, hauteur normale
hauteur = 1, hauteur double
Le 3e paramètre indique le mode:
masque = 0, mode normal, non masqué
masque = 1, mode masqué
Aucun des paramètres n'est obligatoire. En l'absence de l'un des paramètres, l'attribut correspondant n'est pas modifié.
En double hauteur, l'instruction PRINT affiche sur la ligne courante et sur la ligne précédente; le passage à la ligne provoque un double interligne. En double hauteur, il est recommandé de sauter une ligne vide par un PRINT avant le premier affichage.
Les caractères semi-graphiques videotexte ne sont pas affectés par l'instruction ATTRB.
L'exécution de INPUT ou LINE INPUT, après l'affichage éventuel du texte associé, provoque le retour à la taille normale. Les fonctions INKEY$ et INPUT$ ne modifient pas le mode d'affichage.
Le mode masqué provoque l'affichage en caractères noirs sur fond noir. Ces caractères peuvent être démasqués plus tard par l'instruction UNMASK.

Exemple:

```
100 PRINT "DEBUT DU TEXTE"
110 PRINT
120 ATTRB 1,1
130 PRINT "Patience..."
```

AUTO numéro ,pas
commande

Propose automatiquement les numéros de ligne pour l'écriture de programmes.

Le premier nombre indique le numéro de la première ligne à écrire (10 par défaut).
Le deuxième nombre indique l'écart entre les lignes (10 par défaut).
Les deux paramètres sont facultatifs.
Si la ligne proposée existe déjà, le message "Line not empty" s'affiche avant le numéro de ligne.
Le numéro de ligne proposé peut être modifié, le curseur peut être déplacé sur une autre ligne qui peut être modifiée puis validée: la numérotation automatique se cale alors sur le nouveau numéro. On sort de la numérotation automatique par CNT-C ou en validant une ligne sans numéro.

Exemple:

```
AUTO 1000,5
AUTO
```

BACKUP n° de lecteur 1 TO n° de lecteur 2
commande

Recopie entièrement le contenu d'une disquette sur une autre disquette.

Cette commande recopie intégralement la disquette qui se trouve dans le lecteur n° 1 sur la disquette qui se trouve dans le lecteur n° 2.
Avec un seul lecteur, BACKUP n° recopie d'une disquette sur l'autre en plaçant en alternance la disquette "source" et la disquette "destination" dans le lecteur.
BACKUP détruit le programme et les variables présents en mémoire centrale s'il est effectué avec un seul lecteur.

Exemple:

BACKUP 0 TO 1	recopie la disquette 0 sur la disquette 1.
BACKUP 0	lecteur n° 0 seul.
BACKUP 1	lecteur n°1 seul.

BANK

fonction

Donne le numéro de la banque de mémoire courante.

Ce numéro est un nombre compris entre 1 et 6.
A l'initialisation, c'est la banque de rang le plus élevé qui est sélectionnée.

Exemple:

```
? BANK
2
```

BANK numéro

instruction

Sélectionne la banque de mémoire dont on précise le numéro.
Ce numéro doit être compris entre 0 et le nombre maximum de banques de mémoire disponibles (2), ou 6 avec l'extension mémoire 64 K.
Si le numéro est égal à 0, c'est la dernière banque de mémoire existante qui est sélectionnée, comme lors de l'initialisation.
Cette instruction ne concerne que les banques de mémoire situées entre les adresses &HA000 et &HDFFF. Elle a pour effet d'affecter toutes les instructions comportant des références directes à la mémoire: DEF USR, EXEC, CLEAR, LOADM, SAVEM, PEEK, POKE,...

Exemple:

```
BANK 1
```

BEEP

instruction

Produit un son bref.

Cette instruction est équivalente à PRINT CHR$(7); .

Exemple:

```
FOR I=1 TO 100: BEEP: NEXT I
```

BOX (c1,l1)−(c2,l2) caractère,couleur,fond ,inversion
BOX (c1,l1)−(c2,l2) ,couleur
instruction

Trace un rectangle dont les côtés sont parallèles aux bords de l'écran.

Cette instruction fonctionne en mode caractère quand l'argument caractère est présent. Dans ce cas, les coordonnées en colonne (c1,c2) sont comptées de 0 à 39, et en ligne (l1,l2) de 0 à 24.
Elle fonctionne en mode graphique quand l'argument caractère est absent. Dans ce cas, les coordonnées en colonne et en ligne sont comprises entre −32768 et 32767 en Basic 128.

Le rectangle tracé a pour sommets opposés les points situés en (c1,l1) et (c2,l2).

Si le premier point (c1,l1) est omis, il est remplacé par le dernier point du dernier tracé ,c'est-à-dire:
— le 2e sommet dans BOX, BOXF ou LINE,
— le point tracé par PSET,
— le point (0,0) si aucun tracé n'a eu lieu.
L'argument couleur est un nombre.
En mode caractère, l'argument caractère est une chaîne de caractères (variable ou constante) dont le premier caractère est utilisé pour le tracé. Les trois paramètres couleur, fond, inversion sont les mêmes que pour l'instruction COLOR et leur effet est identique.
En mode graphique, le tracé n'a lieu qu'à l'intérieur de la fenêtre definie par WINDOW
En mode caractère et graphique, le type de tracé est fixé par le 3e paramètre de l'instruction CONSOLE.
Les coordonnées du premier point et les arguments couleur, fond et inversion sont facultatifs.

Exemple:

BOX (1,9)−(12,15)"*",3,0	mode caractère
BOX (104,8)−(216,48),2	mode graphique
BOX −(300,70)	

BOXF (c1,l1)–(c2,l2) caractère,couleur,fond ,inversion
BOXF (c1,l1)–(c2,l2) ,couleur
instruction

Trace un rectangle dont les côtés sont parallèles aux bords de l'écran et le remplit du motif indiqué.

En mode caractère,le remplissage se fait avec le caractère indiqué. En mode graphique, le motif de remplissage est celui fixé par l'instruction PATTERN, sinon le remplissage est uniforme.

La syntaxe de l'instruction BOXF est identique à celle de BOX, en mode caractère et en mode graphique.

Exemple:

BOXF (2,2)–(14,14)"+",4,0	mode caractère
BOXF (104,8)–(216,48),2	mode graphique

CDBL(nombre)
fonction

Convertit le nombre en double précision.

Le nombre converti en double précision n'est pas complété uniquement par des 0, ce qui peut conduire à de petites variations. <si l'argument n'est pas un nombre, le message d'erreur "Illegal Function Call" apparaît.

Exemple:

```
?CDBL(454.67)
454.6700134277344
```

CHAIN descripteur de fichier, l1-l2,l3
instruction

Supprime une partie du programme résident en mémoire, fusionne le programme indiqué et poursuit l'exécution à la ligne indiquée.

Cette instruction permet de fusionner une partie de programme et de l'exécuter sans perdre le contenu de toutes les variables.
La portion de programme comprise entre les lignes l1 et l2 est supprimée (même syntaxe que DELETE). Cette option de suppression n'est pas obligatoire.
Puis le programme indiqué est fusionné au programme résident.
Lors de la fusion, les lignes du nouveau programme remplacent les lignes du programme résident, le cas échéant. Le programme à fusionner doit avoir été sauvegardé en binaire (et non pas en ASCII comme dans MERGE).
Si l'option ,l3 est présente, l'exécution se poursuit à la ligne l3, sinon l'exécution reprend à partir de la première ligne du programme.
Seules les variables présentes dans la liste de l'instruction COMMON sont conservées.

Exemple:

`CHAIN "1:SUITE",500-900,100` supprime les lignes 500 à 900, fusionne le programme SUITE de la disquette 1 et poursuit l'exécution en ligne 100

`CHAIN "PG3945",500-` supprime toute la fin à partir de la ligne 500, l'exécution reprend au début

`CHAIN "PASS2"` aucune ligne n'est supprimée, l'exécution reprend au début après la fusion

CHR$(nombre)

fonction

Rend le caractère de code ASCII correspondant au nombre.

Le nombre doit être compris entre 0 et 255.
Cette fonction permet de construire des chaînes de caractères contenant des codes de contrôle. Ainsi la sonnette (code ASCII 7) peut-elle être mise dans une chaîne par CHR$(7).
Les caractères dont le code est supérieur à 127 sont des caractères définis par l'utilisateur au moyen de l'instruction DEFGR$.
CHR$(128+nombre) est équivalent à GR$(nombre).
Les minuscules accentuées peuvent être introduites par CHR$.
La fonction inverse de CHR$ est ASC.

Exemple:

`CHR$(65)`	représente A
`?CHR$(12)`	efface l'écran
`CHR$(22)+CHR$(66)+CHR$(101)`	représente é

CINT(nombre)
fonction

Convertit un nombre réel en un nombre entier.

La conversion se fait par arrondi. Le nombre doit être compris entre −32768 et 32767 sinon l'erreur "Overflow" est déclenchée.

Exemple:

```
?CINT(45.5);CINT(45.4)
 46 45
```

CIRCLE (colonne, ligne) rh, rv, couleur; début, fin
instruction

Trace une portion de cercle ou d'ellipse dont le centre est situé au point (colonne, ligne).

Si rh est présent, cette instruction trace une ellipse de rayon horizontal rh et de rayon vertical rv. Sinon elle trace un cercle de rayon rv.
Si le paramètre couleur n'est pas présent, le tracé se fait dans la couleur courante des caractères.
Si les deux paramètres début et fin sont présents, le tracé ne portera que sur la portion de cercle ou d'ellipse correspondante. Les angles sont comptés en radians; le sens positif étant le sens des aiguilles d'une montre.
Le tracé se fait dans le mode défini par le 3ᵉ paramètre de CONSOLE.

Exemple:

`CIRCLE (100,100) 80,40,5;0,3.1416` demi-ellipse inférieure centrée en (100,100) de rayon horizontal 80, de rayon vertical 40, de couleur magenta

`CIRCLE (160,100), 90, 2` cercle centré au milieu de l'écran, de rayon 90, de couleur verte

CIRCLEF (colonne, ligne) rh, rv, couleur; debut , fin
instruction

Trace une portion de cercle ou d'ellipse dont le centre est situé au point (colonne, ligne) et la remplit du motif courant.

Le motif de remplissage est celui fixé par PATTERN. Par défaut, le remplissage est uniforme. Si seule une portion de cercle ou d'ellipse est tracée, seul le secteur correspondant est rempli.
La syntaxe de CIRCLEF est identique à celle de CIRCLE.

Exemple:

`PATTERN "*" : CIRCLEF (100,100) 80,40,5;0,3.1416` demi-ellipse inférieure centrée en (100,100) de rayon horizontal 80, de rayon vertical 40, de couleur magenta et remplie de caractères " * "

CLEAR volume,adresse1,nombre,adresse2
instruction

Fixe plusieurs paramètres de l'espace mémoire utilisateur et remet à zéro toutes les variables.

Cette instruction supprime toutes les variables et tous les tableaux existants et annule l'effet de l'instruction ON ERROR GOTO GOSUB, ON KEY..., ON INTERVAL.
Le 1er paramètre (volume) est un nombre qui fixe le volume réservé aux chaînes de caractères. Ce volume est fixé par défaut à 300 caractères.

Exemple:

`CLEAR 2000` réserve un espace de 2000 caractères

Le 2e paramètre (adresse1) permet de réserver une zone mémoire dans l'espace correspondant aux banques de mémoires commutables. Cette réservation affecte la banque sélectionnée par l'instruction BANK et les banques de rang supérieur.
L'adresse précisée est un nombre qui est inférieur à &HDFFF.

Exemple:

`BANK 1 : CLEAR ,&HBFFF` réserve l'espace mémoire &HC000 à &HDFFF de la banque 1 et la totalité des banques2 et 3, 4, 5, 6 avec l'extension mémoire 64 K

`BANK 1 : CLEAR ,&H8FFF` réserve l'espace mémoire & H 9000 à

&H9FFF de la zone non commutable et la totalité des banques 2 et 3, 4, 5, 6 avec l'extension mémoire 64 K.

Le 3e paramètre (nombre) fixe le nombre maximum de caractères définis par l'utilisateur. Ce nombre n'est pas modifié jusqu'au prochain CLEAR.

Exemple:

CLEAR ,,10   fixe à 10 le nombre des caractères utilisateur

Le 4e paramètre (adresse2) permet de réserver une zone mémoire dans l'espace non commutable, c'est-à-dire hors banques. L'adresse précisée est un nombre inférieur à &HA000. La réservation ne porte que sur la zone mémoire comprise entre adresse2 + 1 et &H9FFF et n'affecte pas les banques de mémoire.

Exemple:

CLEAR ,,,&H8FFF   réserve l'espace mémoire de &H9000 à &H9FFF

Aucun des paramètres de CLEAR n'est obligatoire.
Les réservations effectuées par une instruction CLEAR ne sont modifiables que par un autre CLEAR.
CLEAR seul supprime toutes les variables mais n'affecte pas les réservations.

CLOSE # n° de canal , # n° de canal

instruction

Ferme le (ou les) fichier(s) dont on indique le numéro de canal.

Cette opération marque la fin du fichier et inscrit dans le catalogue toutes les informations nécessaires. Si aucun numéro de canal n'est précisé, CLOSE ferme tous les fichiers, sauf si une erreur se produit à l'exécution de la fermeture.
Ne pas oublier qu'un arrêt en cours d'écriture ou de lecture par CNT-C ne ferme pas les fichiers. Tous les fichiers doivent avoir été fermés avant de retirer la disquette du lecteur.

Exemple:

CLOSE #1, #3   ferme les fichiers 1 et 3
CLOSE   ferme tous les fichiers ouverts

CLS
instruction

Efface la fenêtre de visualisation définie par CONSOLE et place le curseur en haut à gauche de cette fenêtre.

L'effacement se fait en ramenant tous les points à la couleur du fond. Le même effet peut être obtenu par PRINT CHR$(12).

COLOR forme,fond,inversion
instruction

Fixe la couleur de forme et la couleur de fond pour les affichages et les tracés qui suivent.

Les deux premiers paramètres sont des nombres. Le 1er fixe la couleur de forme, le second la couleur de fond.
Le 3e paramètre est un nombre qui vaut 0 ou 1. Quand il est présent, il provoque une inversion des couleurs de forme et de fond.
Aucun des trois paramètres n'est obligatoire.

Exemple:

COLOR 2,0	caractères et tracés verts sur fond noir
COLOR ,7	sur fond blanc
COLOR ,,1	inverse les couleurs précédentes

COMMON liste de variables
instruction

Permet de transmettre des variables d'un programme à l'autre lors de l'enchaînement de ces programmes par CHAIN.

Les variables de la liste qui suit COMMON sont protégées lors de l'exécution de CHAIN. Le programme fusionné pourra donc les utiliser.
La liste de variables peut contenir des tableaux avec leur dimension; les tableaux correspondants sont alors déclarés, comme dans DIM.
Les variables de la liste sont détruites par CLEAR, RUN, NEW, LOAD, etc.

A l'exécution de COMMON, toutes les variables existantes sont détruites, à l'exception des variables déjà définies par un COMMON. Il est donc préférable de placer les instructions COMMON en début de programme, après un CLEAR éventuel.

Exemple:

`COMMON A,B$,Q(1000)`	protège A,B$ et déclare et protège le tableau Q
`C=3`	
`COMMON C,D`	protège C et D
`?C;D`	
`0 0`	le contenu de C est perdu

CONSOLE ligne haute, ligne basse, tracé, défilement
instruction

Fixe plusieurs paramètres du tracé et de l'affichage.

Tous les paramètres sont optionnels, mais cette instruction ne doit pas se terminer par une virgule et doit comporter au moins un paramètre.
Les deux premiers paramètres fixent la fenêtre d'affichage en mode caractère:
— le 1er indique la ligne supérieure de la fenêtre,
— le 2e indique la ligne inférieure.
Ces deux paramètres sont des nombres compris entre 0 et 24.
Le 3e paramètre précise le mode de tracé graphique:
— s'il est pair, la couleur est modifiée en même temps que la forme,
— s'il est impair, la couleur n'est pas modifiée,
— s'il vaut 0 ou 1, le tracé efface ce qui se trouvait au même endroit sur l'écran (les points du fond du motif sont effacés),
— s'il vaut 2 ou 3, le tracé n'efface pas ce qui se trouvait au même endroit. Un motif agit sur un autre en surimpression.
— s'il vaut 4 ou 5, le tracé inverse les points de la forme existante (OU exclusif). Deux tracés successifs identiques rétablissent l'état initial;

Le 4e paramètre fixe le mode de défilement:
— s'il vaut 0, le défilement a lieu à vitesse normale,
— s'il vaut 1, le défilement a lieu à vitesse lente,
— s'il vaut 2, l'affichage se fait en mode page: quand la fenêtre est pleine, la nouvelle ligne s'affiche en haut de la fenêtre.

Exemples:

CONSOLE 10,20,,2	limite l'affichage aux lignes 10 à 20 incluses avec mode page
CONSOLE ,,1	tracé graphique sans tenir compte de la couleur
CONSOLE ,,4	tracé graphique avec inversion et couleur

CONT
commande

Reprend l'exécution d'un programme après un arrêt.

L'arrêt du programme peut avoir été provoqué par le caractère CNT-C, ou par l'exécution de STOP ou END, ou par une erreur. L'exécution reprend à l'endroit où elle à été interrompue. Si CNT-C a été envoyé au cours de la réponse à une instruction INPUT, à la reprise le message associé apparaît à nouveau et la réponse doit être fournie une deuxième fois.
La commande CONT est utile pour la mise au point de programme en liaison avec l'instruction STOP. Quand l'exécution est suspendue, le contenu des variables peut être affiché et même modifié par une affectation. La commande CONT permet de reprendre l'exécution à l'endroit où elle s'était arrêtée.
La commande CONT ne peut plus fonctionner si le programme a été modifié (message d'erreur Can't Continue).

COPY descripteur de fichier 1 TO descripteur de fichier 2
commande

Permet la copie intégrale d'un fichier (en changeant éventuellement son nom) sur le même support ou sur un autre support.
Avec deux lecteurs de disquettes, COPY permet de copier un fichier de l'un sur l'autre.
Avec un seul lecteur de disquettes, COPY permet de copier un fichier:
— sur la même disquette, en changeant son nom,
— sur une autre disquette, sans changer son nom.
Dans ce cas, on place alternativement la disquette source et la disquette destination dans le lecteur. On n'indique alors dans la

commande que le nom du fichier à recopier (COPY nom de fichier).
En Basic 128, le nom du fichier peut comporter un commentaire.
COPY ne permet pas de transfert d'une disquette vers un autre périphérique.

Exemple:

```
COPY "0:DESS.DAT" TO "1:(4 sept)DESS.DAT"
COPY "MENU.BAS" TO "CHOIX.BAS"
COPY "BUDGET.DAT"
```

COS(nombre)
fonction

Donne le cosinus de l'angle exprimé en radians.

En Basic 128, le résultat est en double précision si l'angle est en double précision.

Exemple:

```
? COS(0.2)
.980067
```

CRUNCH$(chaîne de caractères)
fonction

Analyse la chaîne donnée en argument et rend une chaîne de code évaluable par la fonction EVAL.

Cette fonction permet de coder une chaîne de caractères qui respecte la syntaxe d'une expression du langage Basic. Cette expression, une fois codée par CRUNCH$, peut être évaluée par EVAL.
La chaîne à coder doit être une expression numérique ou une expression chaîne.

Exemple:

```
10 DO
20   LINE INPUT "Expression:";R$
30   PRINT EVAL(CRUNCH$(R$))
40 LOOP
RUN
Expression: 3+4+5
 12
Expression: RIGHT$("DEMAIN",4)
MAIN
...
```

CSNG(nombre)
fonction

Convertit le nombre en simple précision.

La conversion se fait après arrondi.
Les autres fonctions de conversion sont CINT et CDBL.

Exemple:

```
? CSNG(975.341817#)
975.342
```

CSRLIN
fonction

Donne le numéro de la ligne où se trouve le curseur.

Le résultat est un nombre entier compris entre 0 (ligne du haut) et 24 (ligne du bas).
La fonction POS permet de connaître la position du curseur sur la ligne.

Exemple:

`LOCATE 5,8:?CSRLIN` affiche 8 en colonne 5, ligne 8

CVD (variable chaîne de caractères)
fonction

Assure la conversion du contenu de la variable (chaîne de caractères) en un nombre en double précision.

La variable doit contenir des données qui ont été enregistrées par une conversion à l'aide de MKD$( ).
Cette fonction est appliquée le plus souvent à une variable de champ définie dans un FIELD, après lecture d'un enregistrement par GET#.

Exemple:

```
A#=CVD(PRIX$)
```

CVI (variable chaîne de caractères)
fonction

Cette fonction analogue à CVD( ) assure la conversion du contenu de la variable en un nombre entier.

La variable doit contenir des données enregistrées par une conversion à l'aide de MKI$( ).

Exemple:

```
I%=CVI(Q$)
```

CVS (variable chaîne de caractères)
fonction

Cette fonction analogue à CVD( ) assure la conversion du contenu de la variable en un nombre en simple précision.

La variable doit contenir des données enregistrées par une conversion par MKS$( ).

Exemple:

```
J=CVS(TAUX$)
```

DATA liste de constantes
instruction

Cette instruction permet de déclarer une liste de constantes qui pourront être lues par l'instruction READ.

Les instructions DATA ne sont pas exécutables et peuvent se trouver n'importe où dans le programme. L'instruction DATA peut contenir autant de constantes que la longueur de la ligne le permet et il n'y a pas de limite au nombre des instructions DATA contenues dans un programme.
Les constantes de la ligne peuvent être de type entier, réel, double précision ou chaîne de caractères. Les expressions ne peuvent pas figurer dans cette liste.
Les chaînes de caractères doivent être encadrées par des guillemets si elles contiennent un séparateur (virgule, deux-points, ou espaces significatifs en début et en fin). Dans les autres cas, les guillemets peuvent être omis.
Les diverses constantes de la liste sont séparées les unes des autres par une virgule.
L'ensemble des constantes contenues dans les instructions DATA d'un programme peuvent être lues par des instructions READ de façon séquentielle. Le type des constantes lues doit correspondre au type des variables des instructions READ qui reçoivent les valeurs ainsi lues.
Les données d'une même instruction DATA peuvent être lues plusieurs fois si l'instruction RESTORE fait remonter à la ligne de DATA correspondante.

Exemple:

```
100 READ A,B,C#,A$
110 PRINT A,B,C#,A$
...
200 DATA 1,2,3.456789012,"VOS BEAUX YEUX"

RUN
1 2 3.456789012 VOS BEAUX YEUX
```

DEF FN nom (variable ,variable...) = expression
instruction

Définit une nouvelle fonction qui porte sur les variables indiquées entre parenthèses.

Le nom de la fonction suit exactement les mêmes règles qu'un nom de variable. La partie à droite du signe = est l'expression qui définit la fonction. Elle utilise les mêmes variables que celles indiquées dans la partie gauche. Les noms de variables peuvent être identiques à ceux déjà utilisés par ailleurs, sans qu'il y ait d'interaction.
La définition de la fonction doit se faire avant son utilisation. Les fonctions ainsi définies s'utilisent comme les fonctions ordinaires de Basic. On les appelle par leur nom: FN nom (variable, variable,...). Les fonctions définies peuvent donner un résultat numérique ou un résultat chaîne de caractères.

Exemple:

```
DEF FN DIST (X,Y) = SQR (X 2 + Y 2)
A = FNDIST (3,5)
DEF FNCH$ (1) = MID$(STR$(1),2)
A$ = FNCH$ (D$)
```

DEFDBL lettre-lettre, lettre-lettre,...
DEFINT lettre-lettre, lettre-lettre,...
DEFSNG lettre-lettre, lettre-lettre,...
DEFSTR lettre-lettre, lettre-lettre,...

instruction

Déclare le type d'un groupe de variables commençant par la même lettre.

Ces instructions de déclaration affectent toutes les variables qui ne comportent pas de type à la fin du nom, c'est-à-dire qui ne se terminent pas par %, !, # ou $ et dont le nom commence par une lettre qui se trouve dans l'une des plages indiquées.

DEFDBL	déclare le type double précision
DEFINT	déclare le type entier
DEFSNG	déclare le type réel simple précision
DEFSTR	déclare le type chaîne de caractères

En l'absence d'instruction de déclaration de type, ou de caractère marquant le type, toute variable est considérée comme numérique en simple précision.

Les instructions de déclaration de type doivent être exécutées avant l'utilisation des variables qu'elles concernent. Il est préférable de les placer en début de programme.

Exemple:

DEFDBL A-C,W-Z   toutes les variables dont le nom commence par A,B,C,W,X,Y,Z sont de type double précision
DEFINT I,J   les variables dont le nom commence par I ou J sont entières

DEFGR$(n°) = liste de huit nombres
instruction

Définit le caractère utilisateur de n° indiqué.

Le numéro du caractère est un nombre compris entre 0 et 127.
La définition de caractères utilisateur n'est possible que s'ils ont été réservés par une instruction CLEAR précédente (3e paramètre).
Le numéro du caractère ne peut être supérieur au nombre de caractères réservés dans CLEAR moins un.
La définition des caractères se fait par huit segments superposés de huit points chacun.
A chaque point d'un segment on fait correspondre le nombre binaire:
  1 s'il est marqué,
  0 s'il n'est pas marqué.
A chaque segment, on fait correspondre le nombre binaire ainsi obtenu. Sa valeur, en décimal, est donc comprise entre 0 et 255.
Dans la définition d'un caractère, on indique, à droite du signe =, la liste des huit nombres correspondant aux huit segments qui forment le caractère, en commençant par le segment supérieur. Ces huit nombres peuvent être donnés sous forme de constantes, de variables ou d'éléments de tableau.
Le caractère ainsi défini est désigné par GR$(n°) ou par CHR$(128+n°).

Exemple:

```
10 CLEAR ,,2
20 DEFGR$(0) = 128,64,32,16,24,36,66,129
30 DEFGR$(1) = &B1111111, &B10000001, &B10000001, &B10000001, &B10000001,
&B10000001, &B111111111
40 PRINT GR$(0);
50 PRINT GR$(1)
```

DEF USRn = adresse
instruction

Définit la fonction utilisateur en langage machine, de numéro n, commençant à l'adresse mémoire indiquée.

Il peut y avoir dix fonctions utilisateur simultanées, numérotées de 0 par défaut à 9.
L'adresse indique le début de la fonction en langage machine. Une zone mémoire destinée aux fonctions utilisateur peut être réservée par: CLEAR, adresse ou CLEAR,,,adresse.
Le module en langage machine doit se terminer par l'instruction RTS du 6809 et ne doit pas modifier les contenus des registres S et DP.
Le passage des paramètres utilise, à l'appel comme au retour, les registres suivants:

Accumulateur A: 2 nombre entier
(type de la variable) 3 chaîne de caractères
4 simple précision
8 double précision

Index X: 2,X nombre entier
(pointeur sur la valeur) 0,X réel simple ou double précision
0,X descripteur de chaîne de caractères

Accumulateur B: longueur de la chaîne de caractères
Registre U: adresse du 1e octet de la chaîne

Les nombres entiers sont codés sur deux octets (notation en complément à deux), les réels simple précision sur quatre octets (1er octet: exposant, reste: mantisse), les réels double précision sur huit octets (idem).
Le descripteur de chaîne de caractères comprend trois octets: le premier contient la longueur de la chaîne, les deux suivants l'adresse du 1er caractère.
Exemple:

DEF USR2 = &HDF80	définit la fonction numéro 2 à l'adresse &HDF80
W = USR2 (V)	appel de la fonction

DELETE plage de n° de ligne, ligne
commande ou instruction

Supprime des lignes du programme et poursuit l'exécution à la ligne indiquée.

Les lignes du programme à supprimer sont indiquées de la manière suivante:

DELETE	ligne supprime la ligne
DELETE .	supprime la ligne courante
DELETE ligne1-ligne2	supprime toutes les lignes de la ligne1 à la ligne2 incluses
DELETE ligne1-	supprime toutes les lignes de la ligne1 à la fin
DELETE -ligne2	supprime toutes les lignes du début jusqu'à la ligne2
DELETE -	supprime tout le programme

Quand le 2ᵉ paramètre est présent, il indique la ligne à exécuter après la suppression.
Cette possibilité peut servir à éliminer une partie déjà exécutée pour libérer l'espace nécessaire aux variables qui suivent.

Exemple:

DELETE 100–490,50	supprime les lignes 100 à 490 et poursuit l'exécution en ligne 50
DELETE 80	supprime la ligne 80
DELETE -	supprime tout le programme

DENSITY n° de lecteur,densité
instruction

Fixe la densité d'enregistrement pour le lecteur désigné.

Le numéro de lecteur est un nombre compris entre 0 et 3.
La densité est indiquée par:
— 1 pour la simple densité ,
— 2 pour la double densité.

Exemple:

DENSITY 0,2	met le lecteur 0 en double densité

DEVICE "nom de périphérique"
instruction

Détermine le nom du périphérique qui sera pris par défaut dans un descripteur de fichier qui ne le précise pas.

A l'initialisation, le périphérique implicite est le lecteur ϕ.

exemple:

DEVICE "1:" le lecteur 1 devient le périphérique pris par défaut
DEVICE "CASS:"

DIM liste de tableaux
avec tableau = nom de variable (liste d'indices)

instruction

Réserve la place nécessaire aux tableaux à une ou plusieurs dimensions.

Les indices sont des nombres arrondis à l'entier le plus proche qui fixent la valeur maximum de chaque indice du tableau. La valeur minimum que peut prendre un indice est zéro.
L'instruction DIM est exécutable et lors de son exécution, les éléments d'un tableau numérique sont initialisés à la valeur 0 et les éléments d'un tableau de chaînes sont initialisés à la valeur "chaîne vide".
Si, à l'exécution, un indice est en dehors des limites spécifiées dans DIM, l'erreur "Bad Subscript" est détectée.
Il est possible d'utiliser des tableaux sans les déclarer. Dans ce cas, la valeur numérique des indices ne doit pas dépasser 10.

Exemple:

DIM A(15,12) tableau à deux dimensions dont le premier indice peu varier de 0 à 15 et le second de 0 à 12
DIM C$(14) tableau de chaînes de caractères de quinze éléments

DIR "n° de lecteur: nom . suffixe"

instruction

Liste le répertoire de la disquette relatif aux fichiers précisés.

Si le n° de lecteur n'est pas précisé, on obtient le répertoire de la disquette du lecteur par défaut ("0:" sauf modification par DEVICE).
Si le nom n'est pas précisé, la liste porte sur tous les fichiers ayant le

suffixe indiqué. Inversement, si le suffixe n'est pas précisé, la liste porte sur tous les fichiers de ce nom.
Si le nom est incomplet, la liste porte sur tous les fichiers commençant par la racine indiquée.
L'en-tête du répertoire indique la densité de la disquette, le n° du lecteur, son nom s'il existe et le nombre de k-octets libres.

Pour chaque fichier listé, les renseignements du répertoire comprennent six parties:
1. Nom du fichier (huit 8 caractères)

2. Suffixe
BAS: programme Basic
DAT: données
BIN: binaire
MAP: image
TRA: trace

3. Type de fichier
B: programme Basic
D: données Basic
A: programme en assembleur
M: programme en langage machine

4. Type de données
A: ASCII
B: Binaire

5. Taille du fichier en nombre de k-octets

6. Commentaire (huit caractères) s'il existe

L'exécution de DIR pour un périphérique autre qu'un lecteur de disquettes provoque une erreur "Illegal Function Call".

Exemple:

DIR "0:TRICTRAC.BAS" liste le fichier TRICTRAC.BAS dans le répertoire du lecteur 0
DIR "1:.DAT" liste tous les fichiers de données du lecteur 1
DIR "T" liste tous les fichiers commençant par T sur le lecteur courant
DIR liste le répertoire complet du lecteur courant

DIRP "n° de lecteur: nom . suffixe"

instruction

Liste le répertoire relatif aux fichiers précisés sur l'imprimante parallèle.

La syntaxe est identique à celle de DIR.
Le listage est obtenu sur l'imprimante parallèle.

Exemple:

DIRP "2:" liste sur imprimante parallèle tous les fichiers du lecteur 2

DO...LOOP

Instruction générale d'itération.

Toutes les instructions comprises entre DO et LOOP sont répétées. On ne peut sortir normalement de la boucle d'itération que par l'instruction EXIT. L'exécution de EXIT fait sortir de la boucle d'itération; l'exécution se poursuit alors à la première instruction qui suit LOOP.
Les instructions de branchement (GOTO, ON...GOTO et THEN n° de ligne) permettent une sortie à un numéro de ligne différent de la sortie normale. Ce mode de sortie n'est utile que pour les cas exceptionnels (erreur, arrêt du programme,...) car il fait perdre les avantages de la structuration du programme.
Il n'est pas possible de rentrer au milieu d'une boucle DO...LOOP par une instruction GOTO. L'interpréteur Basic, à l'exécution de LOOP, cherche le DO correspondant. Quand le DO correspondant n'a pas été exécuté, le message d'erreur "Loop Without DO" apparaît.
Il est possible de construire des boucles emboîtées en prenant soin de placer la boucle interne entièrement à l'intérieur de la boucle externe.
Dans les cas de plusieurs boucles emboîtées, on peut sortir de plusieurs boucles par une seule instruction EXIT. Le nombre qui suit EXIT indique le nombre de boucles dont il faut sortir.

Exemples:

```
DO: PRINT "BONJOUR" : LOOP        itération infinie
10 DO
20   INPUT A
30   IF A=0 THEN EXIT
40   PRINT A
50 LOOP
60 PRINT "FINI"
```

itération des lignes 20 à 40. Quand la réponse vaut 0, sortie de la boucle

DSKF (n° de lecteur)

fonction

Donne le nombre de k-octets libres sur la disquette située dans le lecteur indiqué.

Le numéro de lecteur est obligatoire.

Exemple:

```
DSKF (0)        volume libre sur la disquette du lecteur 0
```

DSKI$ (n° de lecteur, piste, secteur)

fonction

Effectue la lecture d'un secteur pour lequel on donne le numéro, le numéro de la piste et celui du lecteur où il se trouve.

Le résultat est une chaîne de caractères de 128 octets en simple densité ou de 255 octets en double densité. Le numéro de lecteur est compris entre 0 et 4; le numéro de piste entre 0 et 39, le numéro de secteur entre 1 et 16 inclus. Toute tentative de lecture en dehors de ces valeurs provoque une erreur "Illegal Function Call".

Exemple:

```
DSKI$ (0,20,2)        lecture du secteur 2 de la piste 20 du lecteur 0
```

DSKINI n° de lecteur , facteur d'entrelacement ,nom
commande

Formate la disquette qui se trouve dans le lecteur précisé.

La disquette ne doit pas être protégée en écriture.
Le formatage d'une disquette déjà enregistrée provoque son effacement intégral et toutes les données qu'elles contenait sont perdues.
Si la disquette est défectueuse, le message Input/Output Error apparaît.
Le facteur d'entrelacement, qui n'est ni obligatoire ni très utile, permet de choisir le nombre de secteurs compris entre deux secteurs de numéros consécutifs (implicitement fixé à 7 en l'absence de précision).
Le numéro du lecteur est obligatoire.
Le nom de la disquette, qui n'est pas obligatoire, est une chaîne de huit caractères maximum.
Si l'on a fait au préalable VERIFY ON, l'initialisation a lieu avec vérification et dure beaucoup plus longtemps.

Exemple:

`DSKINI 1,6,"TRAVAIL"` formate la disquette qui se trouve dans le lecteur 1 avec un entrelacement de 6 et nomme cette disquette TRAVAIL
`DSKINI 0` formate la disquette qui se trouve dans le lecteur 0

DSKO$ n° de lecteur, piste, secteur, chaîne de caractères
instruction

Dépose la chaîne de caractères dans le secteur dont on donne le numéro, le numéro de la piste et le numéro du lecteur sur lequel il se trouve.

Le numéro de lecteur doit être compris entre 0 et 4, le numéro de la piste entre 0 et 39, le numéro du secteur entre 1 et 16. La chaîne de caractères doit avoir une taille de 128 caractères maximum en simple densité ou 255 en double densité. Si sa taille est inférieure, le

secteur sera rempli de caractères de code ASCII 0 dans la partie vide après le dernier caractère.

Exemple:

```
DSKO$ 0,18,3,HA$
DSKO$ 0,20,1,"JEUX"      donne le nom JEUX à la disquette du lecteur 0
```

END
instruction

Termine l'exécution d'un programme.

Elle provoque d'abord la fermeture de tous les fichiers ouverts immédiatement au moment de son exécution, puis le passage en mode commande qui se traduit par le message OK sur l'écran.
L'instruction END ne modifie ni le contenu des variables ni les options de CLEAR.
Elle peut être placée n'importe où dans le programme pour terminer l'exécution. Elle n'est pas indispensable à la dernière ligne du programme.

EOF (n° de canal)
fonction

Retourne −1 (VRAI) si la fin du fichier est atteinte ou 0 (FAUX) sinon.

Cette fonction évite l'erreur "Input Past End " dans la lecture d'un fichier.

Exemple:

```
10 ' LECTURE
20 OPEN "I",#1,"DONNEES"
30 DO
40   IF EOF(1) THEN EXIT
50   INPUT#1, A: PRINT A
60 LOOP
```

variables ERL et ERR

ERL et ERR sont des variables particulières qui permettent, lorsqu'une erreur est détectée pendant l'exécution d'un programme, de connaître le numéro de la ligne de l'erreur détectée et le numéro de l'erreur afin de la traiter.
Elles sont réservées à cet usage et toute tentative d'affectation de valeur à ces variables au moyen d'une instruction d'affectation ou d'une instruction de lecture est interdite et détectée comme une erreur de syntaxe.
Lorsqu'une erreur est détectée, deux cas se présentent:
— Aucun traitement d'erreur n'est indiqué; l'exécution du programme est interrompue et le message d'erreur est affiché à l'écran.
— Un traitement d'erreur a été mis en état de veille au moyen de l'instruction ON ERROR. Dans ce cas, la partie de programme chargée du traitement de l'erreur est exécutée. Le numéro de la ligne où s'est produite l'erreur est contenu dans la variable ERL et le numéro de l'erreur est contenu dans ERR. Ainsi les instructions du module de traitement de l'erreur peuvent-elles connaître le type de l'erreur.
La liste de codes d'erreur est fournie en Annexe 7.

Exemple:

```
5 ON ERROR GOTO 100
10 DO
20   INPUT X
30   A=1/X
40   PRINT "L'inverse est ";A
50 LOOP
100 ' module de traitement d'erreur
110 IF ERR=11 THEN PRINT "Donner un nombre plus grand"
120 IF ERR=6 THEN PRINT "Donner un nombre plus petit"
130 RESUME 20
```

ERROR nombre

instruction

Simule une erreur de code existant ou non.

Le nombre est converti en entier par arrondi et donne le numéro de l'erreur.
Cette erreur simulée pourra soit provoquer l'édition du message d'erreur, soit être traitée par le module désigné par ON ERROR.
Si aucune instruction ON ERROR n'est exécutée, le message d'erreur s'affiche ou, s'il s'agit d'un code inconnu de l'interpréteur Basic, le message "Undefined Error".

Exemple:

```
10 S=10
20 T=5
30 ERROR S+T ' simulation de l'erreur 15
RUN
String Too Long in 30
```

EVAL(chaîne de caractères)
fonction

Rend la valeur fournie par l'évaluation de la chaîne codée par CRUNCH$.

Le résultat de l'évaluation est le même que celui obtenu par la même expression dans un programme.
Si la chaîne n'est pas une expression codée par CRUNCH$, le message Syntax Error apparaît.

Exemple:

```
10 LINE INPUT A$
20 ? EVAL (CRUNCH$ (A$))
RUN
? 3^2 + 4^2
25
```

EXEC adresse , liste de paramètres
instruction

Permet d'exécuter un module écrit en langage machine résident en mémoire centrale.

Si l'adresse n'est pas précisée, c'est celle du dernier EXEC ou l'adresse d'exécution du dernier LOADM qui est prise par défaut.
Si l'adresse est comprise entre &HA000 et &HDFFF, c'est la banque de mémoire fixée par BANK qui est adressée ou à défaut la banque de rang le plus élévé.
Des paramètres peuvent être passés au module. Ils sont mentionnés après l'adresse, séparés par des virgules.
On peut les évaluer en appelant, dans le programme binaire, les modules d'analyse de l'interpréteur Basic. L'appel se fait par JSR adresse. Chaque appel évalue un paramètre.
La liste des adresses des modules d'analyse est fournie en Annexe 4.
Le module en langage machine doit se terminer par l'instruction RTS pour provoquer le retour au programme Basic. Les registres S (pointeur de pile) et DP (page directe) ne doivent pas avoir été modifiés.

EXIT nombre
instruction

Instruction de sortie de boucle DO...LOOP ou FOR...NEXT.

L'exécution de cette instruction provoque le branchement à la première instruction qui suit LOOP ou NEXT.
Le nombre indique de combien de boucles il faut sortir. Par défaut, il est pris à 1. Un nombre à 0 ne fait pas sortir de la boucle.

Exemple:

```
10 I=0
20 DO
30   I=I+1: PRINT I
40   IF INKEY$<>"" THEN EXIT
50 LOOP
```

La boucle (incrément d'une variable, affichage de sa valeur) se termine quand on appuie sur une touche

EXP(nombre)
fonction

Retourne l'exponentielle du nombre soit e^nombre.

Rappelons que e vaut 2.712828.
En Basic 128, le résultat est en double précision si l'argument est en double précision.
Le dépassement de capacité est atteint dès que le nombre dépasse 58.02969.

Exemple:

```
? EXP(2)
 7.38906
```

FIELD # n° de canal, longueur 1 AS variable de champ 1 ,longueur 2 AS variable de champ 2..
instruction

Définit l'organisation en champs de l'enregistrement du fichier à accès direct dont on donne le numéro de canal.

Pour chaque champ, on précise sa longueur en caractères et le nom de la variable de champ qui le désigne. Cette variable est toujours une variable chaîne de caractères.
La somme des longueurs ne doit pas dépasser la longueur de l'enregistrement du fichier qui a été fixée au moment de l'ouverture (OPEN). Si la longueur n'a pas été fixée dans OPEN, elle est de 255 caractères (ou 128 en simple densité).
L'instruction FIELD est en général exécutée une fois, après l'ouverture du fichier par OPEN. Tous les enregistrements sont alors lus et écrits suivant ce format. Cependant il est possible d'exécuter des instructions FIELD différentes pour lire des enregistrements de formats différents ou même d'utiliser plusieur FIELD pour lire ou écrire dans le même enregistrement sous des formats différents.
Les variables de champs ainsi définies ne peuvent pas être manipulées comme n'importe quelle variable. Leur contenu est toujours utilisable pour être affiché, transformé ou affecté à une autre variable. Par contre, on ne peut y déposer une valeur que par les instructions LSET, RSET et MID$()=.

Ces variables chaînes de caractères peuvent également recevoir des nombres après que ceux-ci ont été convertis par les fonctions MKI$ (nombres entiers), MKS$ (nombres réels en simple précision) ou MKD$ (nombres en double précision). La taille d'un nombre converti par ces fonctions est de:
— 2 octets s'il est entier
— 4 octets en simple précision
— 8 octets en double précision

Exemple:

FIELD # 2,15 AS NOM$,4 AS Q$ définit, pour le canal 2, deux champs: le premier de quinze caractères et de nom NOM$, le second de quatre caractères et de nom Q$; on pourra affecter un nombre en simple précision à Q$.

FILES nombre de canaux , longueur
instruction

Réserve le nombre de canaux utilisables simultanément dans un programme.

Ce nombre ne peut pas être supérieur à 15.
Chaque canal réservé occupe une zone de 281 octets (ou 153 en simple densité) pour les informations concernant le fichier et la zone de transit des enregistrements.
A l'initialisation, le nombre de canaux réservés est fixé à 2. Si l'instruction FILES comporte une indication de longueur (celle-ci n'est pas obligatoire), cette longueur fixe la somme des longueurs des enregistrements des fichiers à accès direct ouverts simultanément. Elle est fixée à 256 octets à l'initialisation (deux fichiers avec enregistrements de 128 octets).
Par exemple, pour pouvoir ouvrir trois fichiers à accès direct dont les longueurs d'enregistrement sont respectivement 30, 40 et 50 caractères, il faut réserver une longueur d'au moins 120 caractères.
L'instruction FILES remet à zéro toutes les variables existantes. Elle doit donc être exécutée en mode direct ou en tout début de programme.

Exemple:

FILES 3,120

FIX(nombre)
fonction

Rend la partie entière du nombre.

La partie entière est obtenue par troncature. Le résultat est un nombre réel.
Le résultat de FIX est équivalent à celui de INT pour les nombres positifs, il en diffère pour les nombres négatifs.

Exemple:

```
?FIX(3.4),INT(3.4)
 3 3
?FIX(-3.6),INT(-3.6)
 -3  -4
```

FOR...NEXT
FOR variable = valeur initiale TO valeur finale STEP pas
...
NEXT variable, variable

Instruction d'itération permettant de répéter une séquence d'instructions un nombre fixé de fois.
La variable qui suit FOR est appelée variable de contrôle. Elle est numérique de type entier ou simple précision.
A toute instruction FOR doit correspondre une instruction NEXT, mais une instruction NEXT peut correspondre à plusieurs instructions FOR. Dans ce cas, les variables de contrôle de chacune des instructions FOR sont indiquées après NEXT.

Au départ, la variable de contrôle prend la valeur initiale indiquée. Puis l'interpréteur teste si la variable de contrôle est supérieure à la valeur finale.
— Si oui, les instructions de la boucle ne sont pas exécutées et l'exécution se poursuit après NEXT.
— Si non, les instructions comprises entre FOR et NEXT sont exécutées.

La rencontre de NEXT provoque l'incrémentation de la variable de contrôle de la valeur pas, et l'on revient au test précedent.
La boucle d'itération est exécutée jusqu'à ce que la réponse au test soit positive.
Dans le cas d'un pas négatif, l'interpréteur teste si la variable de contrôle est inférieure à la valeur finale et la variable de contrôle est décrémentée de pas.
Si l'option STEP n'est pas précisée, lincrément par défaut est 1.
En Basic 128, il est possible de sortir de la boucle par l'instruction EXIT. L'exécution se poursuit alors après NEXT, mais en général, la variable de contrôle n'a pas atteint la valeur finale.

Boucles emboîtées

Les boucles FOR peuvent être emboîtées, c'est-à-dire qu'une boucle FOR peut contenir une autre boucle FOR utilisant une variable de contrôle différente.
Si plusieurs boucles ont la même instruction terminale, il est possible d'utiliser un NEXT multiple pour toutes ces boucles:
NEXT variable1,variable2,...
Si la variable de contrôle est omise, l'instruction NEXT correspond à la boucle FOR la plus proche qui n'est pas appariée.

Exemples:

```
10 K=10
20 FOR I=1 TO K STEP 2
30 K=K+10
40 PRINT I:K
50 NEXT I
```

la valeur finale de la variable de contrôle reste 10 bien que K ait été modifiée en cours d'exécution.

```
10 I=0: X=0
20 FOR I=3 TO 2
30 X=X+1
40 NEXT I
50 PRINT I;X
```

la boucle n'a pas été exécutée et la variable de contrôle conserve la valeur initiale (3).

```
10 DIM A(20,10)
20 FOR I=1 TO 20
30 FOR J=1 TO 10
40 A(I,J)=1
50 NEXT J,I
```

deux boucles emboîtées pour l'initialisation d'un tableau.

```
100 FOR I=1 TO 20
110 IF A(I)=0 THEN EXIT
120 NEXT I
130 IF I=21 THEN PRINT "PAS D'ELEMENT NUL"
```

recherche d'un élément nul dans un tableau et sortie par EXIT.

FRE(donnée)

fonction

Permet de connaître l'espace mémoire libre.

Quatre informations sont disponibles:
1. FRE(0) donne la place totale disponible en mémoire.
2. FRE(1) donne la place disponible dans la zone de travail de l'interpréteur (entre &H6100 et &H9FFF).
3. FRE(2) donne la place disponible pour le programme et les données (entre &HA000 et &HDFFF dans les banques de mémoire).
4. FRE(A$) donne la place disponible pour les chaînes. Cet espace est fixé à 300 caractères à l'initialisation.

Noter que FRE(0) = FRE(1) + FRE(2)

Exemple:

```
10?FRE(0);FRE(1);FRE(2);FRE(A$)
20 X$="0123456789"
30?FRE(0);FRE(1);FRE(2);FRE(A$)
```

FWD nombre *
instruction

Déplace une tortue dans sa direction de la quantité indiquée.

Le nombre doit être compris entre −255 et 255.
Si le nombre est négatif, la tortue recule.
Si la tortue est visible, elle est effacée de son ancienne position et affichée dans la nouvelle.
Si la trace est active, le déplacement est tracé sur l'écran.
Toutes les modifications de taille, de direction ou de rotation sont prises en compte avant le déplacement.
La tortue déplacée est la dernière désignée par TURTLE.
Si aucune instruction TURTLE n'a été exécutée, l'erreur Illegal Function Call est déclenchée.

Exemple:

`FWD 30` déplace la tortue de trente points dans sa direction

GET (colonne 1, ligne 1) - (colonne 2, ligne 2), élément de tableau *
instruction

Mémorise dans un tableau numérique la portion d'image graphique contenue dans le rectangle défini par les deux sommets (colonne 1, ligne 1) et (colonne 2, ligne 2).

La portion d'image est obligatoirement composée d'un nombre entier de caractères. Les coordonnées des deux sommets sont donc données en caractères: colonnes de 0 à 39 et lignes de 0 à 24.
Si le deuxième sommet n'est pas précisé, c'est le sommet en bas à droite qui est pris par défaut (39,24). Le tableau numérique doit avoir été déclaré au préalable et il doit être de type entier.
Plusieurs images peuvent être mémorisées dans un seul tableau. Elles sont systématiquement compactées et rangées depuis le bas du tableau, c'est-à-dire de l'indice le plus élevé vers les indices plus faibles.

L'élément de tableau désigné dans l'instruction GET est l'élément d'indice le plus élevé pour cette image. L'image est mémorisée à partir de l'élément précédent. Après l'exécution, l'élément désigné contient la valeur de l'indice du premier élément libre du tableau. C'est cet élément qui pourra servir pour une nouvelle instruction GET ou LOADP avec le même tableau.
L'image est compactée dans le tableau, le volume qu'elle occupe est dépendant de son contenu.
Si le tableau est trop petit, une erreur "Out of Memory" est détectée.
La portion d'image ainsi mémorisée peut être restituée en un autre endroit de l'écran par PUT.
Elle peut être conservée en fichier par l'instruction SAVEP.

Le mode de compactage est compatible avec les outils COLORPAINT et PRAXITEL.

Exemple:

DIM T%(1000) déclare le tableau de réception
GET (0,0)−(9,9),T%(1000) mémorise la portion d'image à partir de la fin du tableau
? T%(1000) contient le premier élément libre
817 c'est le 817
GET (0,10)−(9,19),T%(817) mémorise une deuxième portion à partir du premier élément libre
? T%(817) nouveau premier élément libre
760 c'est le 760 ... et ainsi de suite

GET # n° de canal, n° d'enregistrement
instruction

Transfère un enregistrement d'un fichier à accès direct (dont on donne le numéro de canal) vers la zone tampon en mémoire centrale.

La récupération du contenu de l'enregistrement peut alors se faire par INPUT#, ou directement dans les variables de champ si un FIELD a été exécuté au préalable.
Si le numéro d'enregistrement n'est pas précisé, GET lit l'enregistrement suivant du fichier ou le premier si aucun enregistrement n'a encore été lu ou écrit.

Exemple:

GET#2,10   lit le 10e enregistrement du fichier n° 2

GOSUB numéro de ligne
instruction

Permet le branchement à un sous-programme.

L'exécution d'un sous-programme appelé par GOSUB débute à la première instruction de la ligne indiquée par son numéro. L'exécution du sous-programme se termine par l'instruction RETURN, elle provoque le retour à l'instruction qui suit immédiatement le GOSUB appelant.
Un sous-programme peut avoir plusieurs points d'entrée. De même, il peut contenir plusieurs instructions RETURN.
Un sous-programme peut appeler un autre sous-programme et ainsi de suite jusqu'à encombrement complet de la mémoire. Un sous-programme peut s'appeler lui-même.
Une tentative d'exécution de GOSUB vers une ligne inexistante est détectée par l'erreur "Undefined Line".
Un sous-programme peut être placé n'importe où dans un programme, mais il est préférable de le séparer du programme principal par une instruction END.

Exemple:

```
10 GOSUB 100
20 PRINT "1er retour"
30 GOSUB 100
40 PRINT "2e retour"
50 END
100 PRINT "Sous-programme"
110 RETURN
RUN
```

```
10 INPUT N
20 GOSUB 100
30 PRINT "Factorielle:";F
40 END
100 F=1
110 IF N=1 THEN RETURN
120 F=F*N: N=N-1: GOSUB 110
130 RETURN
```

calcul récursif de factorielle N

GOTO numéro de ligne

instruction

Permet le branchement inconditionnel à une ligne de programme.

Si la ligne désignée contient une instruction exécutable, celle-ci sera exécutée, puis les instructions suivantes.
Si la ligne désignée ne contient pas d'instruction exécutable (REM ou DATA), l'exécution se poursuit après cette ligne.
Si la ligne désignée n'existe pas, l'erreur "Undefined Line" est détectée.
En mode d'exécution directe, cette instruction permet de poursuivre au numéro indiqué l'exécution d'un programme interrompu. A la différence de "RUN numéro de ligne", elle n'initialise pas les variables et ne ferme pas les fichiers ouverts.
L'écriture d'une boucle est plus facile avec DO...LOOP.
Exemple:

```
10 PRINT "JE BOUCLE...INFINIMENT"
20 GOTO 10
```

boucle infine avec GOTO

GR$(nombre)

fonction

Rend le caractère utilisateur de numéro indiqué.

Les caractères utilisateurs sont numérotés de 0 à 127. Ils doivent avoir été déclarés au préalable par CLEAR (3e paramètre) et définis par DEFGR$.

GR$(i) est équivalent à CHR$(128+i).

Exemple:

?GR$(23)   affiche le caractère utilisateur n° 23

HEAD
fonction

Rend la direction de la tortue active.

Le résultat est un nombre compris entre 0 et 255.
La tortue active est la dernière fixée par TURTLE.

Exemple:

```
? HEAD
62
```

HEAD TO nombre
instruction

Fixe ou modifie la direction du mouvement de la tortue active.

Si l'option TO est présente, la direction est fixée de façon absolue, sinon elle est modifiée de façon relative.
Le nombre indiquant la direction est compris entre −255 et 255. S'il est plus grand, seul le reste de la division par 256 (modulo) est pris en compte.
Un nombre positif signifie vers la droite, un nombre négatif signifie vers la gauche.
La tortue active est la dernière désignée par TURTLE.

Exemple:

ROT 64   modifie la direction de 90° vers la droite
ROT 320   idem
ROT TO −32   fixe la direction à 45° vers la gauche

HEX$ (nombre)
fonction

Donne une chaîne de caractères représentant le nombre dans la base hexadécimale (16).

Si l'argument n'est pas entier, il est tronqué. Sa valeur doit être comprise entre −65536 et 65535.

Exemple:

```
? HEX$ (65535)
FFFF
```

IF condition GOTO n° de ligne ELSE n° de ligne THEN instructions instructions

instruction

Permet des branchements conditionnels.

L'exécution de cette instruction se fait de la manière suivante:
la condition est d'abord évaluée.
— Si le résultat est VRAI, alors les instructions qui suivent THEN sont exécutées et, si ELSE est présent, les instructions qui le suivent sur la ligne sont ignorées.
— Si le résultat est FAUX, les instructions qui suivent ELSE sont exécutées ou bien, si ELSE est absent, l'exécution passe à la ligne suivante du programme.
Dans chaque cas, si la liste d'instructions est remplacée par un numéro de ligne, il y a branchement à la ligne indiquée.

Exemple:

```
IF B*B-4*A*C < 0 GOTO 500
```
si négatif, branchement en ligne 500

instructions IF emboîtées

Il est possible de faire figurer à l'intérieur d'une instruction IF une ou plusieurs instructions IF dans la limite d'une ligne logique.
L'interprétation de IF emboîtés est la suivante:
— chaque ELSE est associé au THEN le plus proche qui n'est pas associé à un autre ELSE.

Exemple:

```
IF A<B THEN IF B<C THEN PRINT "A<C" ELSE PRINT "B>=C"
IF A<B THEN IF B<C THEN PRINT "A<C" ELSE PRINT "B>=C" ELSE PRINT
"A>=B"
```

Remarque:
Les tests d'égalité des nombres réels ne sont pas toujours exacts puisqu'ils dépendent de la précision. Il faut donc leur préférer le test d'une différence minimum.

Exemple:

```
IF ABS(A-B)<1E-5 THEN PRINT "A et B égaux"
```

INKEY$
fonction

Retourne le dernier caractère tapé au clavier ou une chaîne vide si aucun caractère n'a été tapé.

Cette fonction ne renvoie pas d'écho sur l'écran.
Tous les caractères, y compris les caractères de contrôle, sauf CNT-C et STOP, sont pris en compte par INKEY$. En particulier. l'appui sur ENTREE rend le caractère Retour Chariot.

Exemple:

```
10 FOR I=1 TO 1000: NEXT I
20 A$=INKEY$
30 PRINT "CARACTERE:";A$
```

INPEN variable colonne, variable ligne
INPUTPEN variable colonne, variable ligne
instructions

Permettent de lire les coordonnées du point de l'écran visé par le crayon optique.

INPEN effectue immédiatement la lecture, INPUTPEN attend que le contact du crayon optique soit fermé pour effectuer la lecture.
La première variable reçoit la coordonnée horizontale (entre 0 et 319), la seconde variable reçoit la coordonnée verticale (entre 0 et 199).
Quand la mesure ne peut pas être effectuée, les deux variables reçoivent −1. Cette situation se présente en cas de:
— mauvais réglage du crayon optique,
— luminosité insuffisante du point visé (noir ou rouge par exemple),
— visée d'un point hors de la zone utile de l'écran.

Exemple:

```
100 DO
110   INPUTPEN C,L
120   IF C<>-1 THEN EXIT
130 LOOP
140 ? C,L
```

INPUT chaîne de caractères, liste de variables
INPUT chaîne de caractères; liste de variables

instruction

Instruction d'entrée de données au clavier.

Le message contenu dans chaîne de caractères est affiché (s'il existe).
Il est suivi d'un point d'interrogation et d'un espace si le délimiteur est un point-virgule. Le point d'interrogation n'est pas affiché si le délimiteur est une virgule.
L'affichage des caractères suivants est forcé à la taille normale.
L'utilisateur doit alors introduire les données attendues par la liste de variables. Ces données doivent être de même type que les variables de la liste, sinon le message "Redo from start" est affiché.
Les différentes données sont séparées par des virgules. L'introduction d'une chaîne doit être encadrée par des guillemets si elle contient une virgule ou des espaces significatifs au début ou à la fin.
Les variables peuvent être des éléments de tableaux mais pas des tableaux.

INPUT ne peut être utilisé en mode direct.

Exemple:

```
10 INPUT "JOUR, MOIS, ANNEE";J%,M$,AN%
```

INPUT# n° de canal, liste de variables

instruction

Lit le fichier dont on précise le n° de canal et affecte les valeurs lues aux variables mentionnées dans la liste.

Le type des données (numérique ou chaîne de caractères) doit correspondre au type des variables. Cette instruction obéit aux mêmes règles que INPUT à partir du clavier.
Si les variables sont plus nombreuses que les données, l'erreur "Input Past End" est déclenchée. On s'en prémunit en utilisant la fonction EOF.

Exemple:

`INPUT#2, A$, N` lit les données dans A$ et N successivement

INPUT$ (nombre de caractères, # n° de canal)

fonction

Retourne une chaîne de caractères qui contient le nombre de caractères lus à la suite dans le fichier indiqué.

Cette fonction permet de lire un nombre déterminé de caractères sur le canal considéré. Elle permet de connaître de façon exacte (au caractère près) le contenu d'un fichier.
Si le n° de canal n'est pas précisé, la lecture se fait à partir du clavier sans écho à l'écran.
CNT-C et STOP sont transmis et n'interrompent pas l'exécution.
Au clavier, cette instruction permet d'éviter la frappe de la touche ENTREE.

Exemple:

`INPUT$ (3,#1)` lecture de 3 caractères sur le fichier 1

INPUTTURTLE variable colonne, variable ligne
instruction

Permet de connaître les coordonnées de la tortue active.

La première variable reçoit le n° de colonne, la seconde le n° de ligne. Les valeurs reçues s'étendent de −32768 à 32767.
La tortue active est la dernère désignée par TURTLE.

Exemple:

```
10 TURTLE 1
20 DO
30  FWD 10: HEAD 16
40  INPUTTURTLE C,L: LOCATE 0,0: PRINT C,L
50 LOOP
```

A chaque déplacement, la ligne 40 lit les coordonnées de la tortue et les affiche.

INPUTWAIT n° de ligne; durée, chaîne de caractères, liste de variables
INPUTWAIT n° de ligne; durée, chaîne de caractères; liste de variables
instruction

Permet de lire des données à partir du clavier, en accordant un temps limité pour la réponse.

Le premier paramètre indique la ligne à exécuter si le temps est dépassé.
Le deuxième paramètre indique en secondes le temps maximum autorisé pour taper la réponse.

Le reste de l'instruction est identique à INPUT. En particulier, la chaîne de caractères doit être suivie d'un point-virgule si l'on veut qu'un point d'interrogation soit affiché.
La forme réduite:

INPUTWAIT n° de ligne;durée,"",
permet d'obtenir une temporisation suivie d'un branchement.

Exemple:

```
10 PRINT "DEBUT"
20 DO
30   INPUTWAIT 100;10,"Valeur de A";A
40   PRINT A
50 LOOP
60 END
100 PRINT "Trop lent"
110 GOTO 10
```

attend 10 secondes la réponse

INPUTWAIT 900;5,"", temporise 5 secondes et branche en ligne 900

INSTR (position, chaîne 1, chaîne 2)
fonction

Rend la position de la chaîne 2 dans la chaîne 1.

Si le premier paramètre est présent, la recherche se fait à partir de la position indiquée, sinon elle se fait à partir du premier caractère de chaîne 1.
Le résultat de la fonction INSTR est 0:
— si la chaîne 2 ne figure pas dans la chaîne 1,
— si la chaîne 1 est vide,
— si la position de départ est supérieure à la longueur de la chaîne 1.
Si la chaîne 2 est vide, le résultat de la fonction est la position de départ (1 par défaut).
Si la chaîne 2 est présente dans la chaîne 1, la fonction INSTR rend sa position.

Exemple:

```
? INSTR("IL FAIT BEAU","FAIT")
 4
? INSTR(6,"IL FAIT BEAU","FAIT")
 0
? INSTR(3,"IL FAIT BEAU","")
 3
```

INT (nombre)
fonction

Retourne le plus grand entier inférieur ou égal au nombre précisé.

Le résultat est un nombre réel. L'argument peut être entier, réel ou double précision.

Exemple:

```
?INT(76.75)
 76
?INT(-76.1)
 -77
```

Pour arrondir au nombre entier le plus proche, il faut prendre: INT(X+.5)

INTERVAL ON
INTERVAL OFF
instructions

Permettent de déclencher ou d'arrêter le compteur de temps défini dans l'instruction ON INTERVAL.

KILL descripteur de fichier
commande

Supprime le fichier en question, c'est-à-dire supprime ses références dans le répertoire et libère la place qu'il occupe sur la disquette.

Dans la description du fichier, le nom du périphérique n'est pas obligatoire mais le suffixe est obligatoire. Le périphérique pris par défaut est le lecteur 0 ou celui fixé par DEVICE. L'instruction KILL ne fonctionne que pour un fichier sur disquette. Si le fichier n'existe pas, le message "File Not Found" apparaît.

Exemple:

`KILL "1:BONJOUR.BAS"` supprime le fichier BONJOUR.BAS de la disquette 1

LEFT$ (chaîne , longueur)
fonction

Donne la partie gauche de la chaîne sur la longueur précisée.

Si la longueur est nulle, le résultat est la chaîne vide.
Si la longueur demandée est supérieure à la longueur de la chaîne, le résultat est la chaîne entière.

Exemple:

```
?LEFT$("MELODRAME",4)
 MELO
```

LEN (chaîne)
fonction

Donne la longueur de la chaîne de carctères.

Tous les caractères de la chaîne sont comptés.
Il ne faut pas oublier que les minuscules accentuées et le ç sont codés sur trois caractères.

Exemple:

```
?LEN("TROIS")
5
?LEN("été")
7
```

LET variable = expression
instruction

Permet d'affecter une valeur à une variable.

Le mot clé LET est facultatif.
La variable et l'expression doivent être de même type, sinon l'erreur "Type Mismatch" est détectée.

Exemple:

```
10 D=12
20 SOMME = D+20
30 A$="BELLE"
40 B$=A$+" VUE"
```

LINE (c1,l1)–(c2,l2) caractère,couleur,fond ,inversion
LINE (c1,l1)–(c2,l2),couleur
instruction

Trace une ligne entre les deux points désignés.

Cette instruction fonctionne en mode caractère quand l'argument caractère est présent.
Sa syntaxe est identique à celle de BOX.

Exemple:

LINE (1,9)–(15,15)"+",3,0 trace une ligne de + entre les deux points de couleur 3 et de fond 0
LINE (100,60)–(163,96),2 trace une ligne de couleur 2 entre les deux points graphiques
LINE –(200,200) la ligne part du dernier point trace

LINE INPUT chaîne de caractères; variable chaîne
LINE INPUT chaîne de caractères, variable chaîne
instruction

Permet d'introduire dans une variable chaîne une séquence d'au plus 255 caractères quelconques.

La fin de la réponse est marquée par la touche ENTREE. Tous les caractères frappés, sauf le caratère Retour Chariot sont affectés à la variable chaîne.
La présence de la chaîne de caractères provoque l'affichage du message qu'elle contient.
Que la chaîne soit suivie d'une virgule ou d'un point-virgule, aucun point d'interrogation n'est affiché.
Cette instruction est très pratique pour lire une ligne dont ne connaît pas la structure.

Exemple:

```
10 LINE INPUT "QUOI?";A$
20 PRINT A$
```

LINE INPUT # n° de canal, variable chaîne
instruction

Lit le prochain enregistrement du fichier dans une variable chaîne de caractères en prenant tous les caractères.

Exemple:

```
LINE INPUT #2, X$
```

LIST descripteur de fichier, n° ligne 1 - n° ligne 2
commande

Liste le programme contenu en mémoire, sur l'écran ou sur le fichier désigné.

Les lignes du programme à lister sont indiquées de la manière suivante:

LIST	ligne liste la ligne
LIST .	liste la ligne courante
LIST ligne1-ligne2	liste toutes les lignes de la ligne1 à la ligne2
LIST ligne1-	liste toutes les lignes de la ligne1 à la fin
LIST -ligne2	liste toutes les lignes du début jusqu'à la ligne2

Si le fichier est un fichier sur disque, LIST est équivalent à SAVE avec l'option ,A.
Si aucun fichier n'est précisé, le listage se fait sur l'écran.

Exemple:

LIST "LPRT:(80)", 1000-1990 liste sur l'imprimante parallèle de la ligne 1000 à la ligne 1990
LIST "1:PG3.BAS" liste tout le programme sur le fichier PG3.BAS
LIST liste tout le programme à l'écran

LOAD descripteur de fichier, R
commande

Charge en mémoire le programme dont on a précisé le nom et le support.

La commande LOAD ferme tous les fichiers ouverts, détruit le programme en mémoire puis effectue le chargement proprement dit. Toutes les variables sont détruites.
Avec l'option ,R, le programme chargé est aussitôt exécuté et les fichiers déjà ouverts avant le chargement restent ouverts. Le périphérique pris par défaut est le lecteur 0, ou celui fixé par DEVICE.

Exemple:

`LOAD "2: OTHELLO",R` charge le programme OTHELLO de la disquette 2, laisse les fichiers ouverts et lance l'exécution
`LOAD "TICTAC"` charge le programme TICTAC

LOADM descripteur de fichier, translation ,R
instruction

Charge une zone mémoire en binaire à partir d'un fichier constitué par SAVEM ou par l'assembleur.

Le chargement s'effectue dans la banque courante.
Si une translation est indiquée, les données ou le programme sont décalés de cette translation. La translation décale aussi l'adresse d'exécution.
S'il s'agit d'un programme, l'option R en provoque l'exécution immédiate.

Exemple:

`LOADM "HORLOGE", &H200,R` charge HORLOGE.BIN, le décale de 512 octets et lance son exécution à l'adresse donnée dans SAVEM + 512
`LOADM "FORME"` charge FORME.BIN

LOADP descripteur de fichier, élément de tableau
instruction

Charge dans le tableau l'image contenue dans le fichier spécifié.

Le résultat obtenu est similaire à celui de l'instruction GET.

Le tableau numérique doit avoir été déclaré au préalable et il doit être de type entier.
Plusieurs images peuvent être mémorisées dans un seul tableau. Elles sont systématiquement compactées et rangées depuis le bas du tableau, c'est-à-dire de l'indice le plus élevé vers les indices plus faibles.
Si le suffixe du fichier est absent, .MAP est pris par défaut.

Exemple:

`LOADP "0:MONALISA",IM%(1000)` charge le fichier image MONALISA.MAP du lecteur 0 dans le tableau IM% à partir de l'élément 999. Si l'image occupe 400 octets, après le chargement, IM%(1000) contient 799.

LOC (n° de canal)
fonction

Indique où en est la lecture (ou l'écriture) dans le fichier ouvert sous le n° de canal considéré.

Si le fichier est à accès séquentiel, LOC indique le nombre de secteurs écrits ou lus depuis l'ouverture du fichier.
Si le fichier est à accès direct, LOC indique le numéro de l'enregistrement qui suit celui qui vient d'être lu par GET# ou écrit par PUT#.

Exemple:

`LOC (2)` position de la lecture ou de l'écriture en cours sur le fichier 2

LOCATE colonne, ligne, curseur
instruction

Fixe la position du curseur sur l'écran en mode caractère.
Le premier paramètre indique la colonne et doit être compris entre 0 et 39. Le second indique la ligne et doit être compris entre 0 et 24.
Le troisième paramètre rend le curseur invisible (valeur 0) ou visible (valeur 1).

Les trois paramètres sont facultatifs en Basic 128.
La position du curseur par LOCATE peut se faire en dehors de la fenêtre d'affichage définie par CONSOLE.

Exemple:

```
LOCATE 10,15: PRINT "BONSOIR"
```

LOF (n° de canal)
fonction

Donne la position du dernier enregistrement dans le fichier ouvert sous le n° de canal considéré.

Si le fichier est à accès séquentiel, LOF indique le numéro du dernier secteur du fichier. Si le fichier est à accès direct, LOF indique le numéro du dernier enregistrement.

Exemple:

`LOF (1)` position du dernier enregistrement du fichier 1

LOG(nombre)
fonction

Donne le logarithme népérien du nombre (base e).

En Basic 128, le résultat est en double précision si l'argument est en double précision.

Exemple:

```
?LOG(5)
 1.60944
```

LOOP
instruction

Marque la fin d'une boucle et déroute l'exécution au DO correspondant.

(Voir l'instruction DO.)

LSET variable chaîne = chaîne de caractères
instruction

Dépose une chaîne de caractères dans une variable chaîne en cadrant à gauche.

La chaîne de caractères est déposée en commençant par la gauche. Si sa longueur est plus petite que la taille de la variable, le reste est complété à droite par des espaces; si sa longueur est trop grande, la fin de la chaîne est éliminée. Les nombres peuvent être déposés dans une variable chaîne après avoir été convertis par:
MKI$( )   pour un entier
MKS$( )   pour un nombre en simple précision
MKD$( )   pour un nombre en double précision
Cette instruction est utilisée le plus souvent pour déposer des données dans une variable de champ définie par FIELD.

Exemple:

`LSET ID$ = NOM$ + PR$`   dépose dans ID$ la concaténation de NOM$ et PR$
`LSETQ$ = MKS$(QU)`   dépose la valeur numérique de QU après conversion

MAX(liste de nombres)
fonction

Retourne le maximum de la liste.

Les nombres peuvent être de précisions différentes.

Exemple:

```
?MAX(10,12,5.6#)
 12
```

MERGE descripteur de fichier, R
commande

Charge en mémoire le programme indiqué et l'ajoute au programme résidant.

Les lignes du programme chargé sont classées avec celles du programme résidant et les recouvrent si elles sont identiques. Le programme à charger doit avoir été sauvegardé avec l'option A. L'exécution du programme global est lancée à la fin du chargement avec l'option R.

Exemple:

`MERGE "1:TEST",R` charge le programme TEST de la disquette 1, le fusionne avec le programme en mémoire et lance l'exécution.
`MERGE "MENU"` fusionne le programme MENU au programme résident

MID$ (chaîne, position, longueur)
fonction

Extrait une sous-chaîne de longueur donnée d'une chaîne donnée.

La position est un nombre compris entre 1 et 255, la longueur est un nombre compris entre 0 et 255.
Si la longueur est trop grande ou si ce paramètre est absent, la sous-chaîne va jusqu'à la fin de la chaîne.
Si la position supérieure à la longueur de la chaîne, le résultat est une chaîne vide.

Exemple:

```
?MID$("PATATRAC",3,2)
TA
?MID$("PATATRAC",5)
TRAC
```

MID$ (variable chaîne , début , longueur) = chaîne de caractères
instruction

Permet de remplacer une partie d'une variable chaîne par une autre chaîne.

La longueur de la variable chaîne reste inchangée dans tous les cas.
Le 2e paramètre indique la position du 1er caractère à remplacer dans la variable.
Le 3e paramètre, facultatif, indique la longueur de la partie à remplacer.
— Si la chaîne de remplacement est plus longue que la partie à remplacer, seul le début de cette chaîne est utilisé.

Exemple:

```
10 A$="MICRO-ORDINATEUR"
20 MID$(A$,6,7)="PROCESSEURS"
30 PRINT A$
RUN
MICROPROCESSEUR
```

— Si la longueur n'est pas précisée, le remplacement se fait jusqu'à la fin de la variable, si la chaîne modifiante est assez longue.

Exemple:

```
10 A$="SAINT-LOUIS"
20 MID$(A$,7)="DENISE"
30 PRINT A$
RUN
SAINT-DENIS
```

Cette instruction peut être utilisée pour modifier la valeur d'une variable de champ définie dans FIELD.

MIN(liste de nombres)
fonction

Retourne le minimum de la liste.

Les nombres peuvent être de précisions différentes.

Exemple:

```
?MIN(12,3.4,5.6)
 3.4
```

MKD$ (nombre)
fonction

Assure la conversion d'un nombre en double précision en chaîne de caractères, pour qu'il puisse être déposé par LSET ou RSET dans une variable chaîne.

La longueur de la chaîne de caractères obtenue est de huit octets. Cette chaîne de caractères est codée et non lisible directement.

Exemple:

```
MKD$ (TOTAL #)
MKD$ (1.2345678901234)
```

MKI$ (nombre)
fonction

Assure la conversion d'un nombre entier en chaîne de caractères, pour qu'il puisse être déposé par LSET ou RSET dans une variable chaîne.

La longueur de la chaîne obtenue est de deux octets

Exemple:

```
MKI$ (Q%)
MKI$ (–12536)
```

MKS$ (nombre)
fonction

Assure la conversion d'un nombre en simple précision en chaîne de caractères pour qu'il puisse être déposé par LSET ou RSET dans une variable chaîne.
La longueur de la chaîne obtenue est de quatre octets

Exemple:

```
MKS$ (X)
MKS$ (6.023 E + 23)
```

MOTOR ON / MOTOR OFF
instructions

Comme son nom l'indique, met en marche le moteur du magnétophone à cassettes ou l'arrête.

Exemple :

```
MOTORON
```

NAME descripteur de fichier 1 AS descripteur de fichier 2.
commande

Permet de changer le nom d'un fichier sur disquette.

Descripteur du fichier 1 comporte le nom du fichier à renommer. Descripteur de fichier 2 contient le nouveau nom du fichier. Le nouveau nom ne doit pas correspondre à un fichier déjà existant. En Basic 128, un commentaire peut être indiqué avec le nom du fichier.

Exemple:

```
NAME "1:TIC.BAS" AS "1:TAC.BAS"
NAME "TEXTE.DAT" AS "(notes)OLDTXT.DAT"
```

NEXT liste de variables
instruction

Incrémente ou décrémente les variables de contrôle d'une ou plusieurs boucles FOR...NEXT.

(Pour le fonctionnement de cette instruction, voir FOR.)

NEW
commande

Efface le programme présent en mémoire et détruit toutes les variables, y compris celles réservées dans COMMON.

Cette commande ne ferme pas les fichiers ouverts et ne modifie pas les options fixées au préalable par CLEAR.

Exemple:

```
NEW
```

OCT$(nombre)
fonction

Rend une chaîne de caractères exprimant le nombre en octal (base 8).

Si l'argument n'est pas entier, il est tronqué.

Exemple:

```
?OCT$(30)
36
```

ON ERROR GOTO n° de ligne
instruction

Permet le branchement à la ligne indiquée si, dans le déroulement de l'exécution d'un programme, une erreur est détectée.

La séquence de traitement des erreurs est alors effectuée à partir de la ligne indiquée et les interruptions par ON INTERVAL sont suspendues.
Cette séquence doit se terminer normalement par RESUME.
L'instruction ON ERROR GOTO est placée soit en début de programme, soit au début des séquences où une erreur est susceptible de se produire.
ON ERROR GOTO 0 annule l'effet de ON ERROR GOTO n° et ramène à l'état normal: message d'erreur et arrêt de l'exécution.
L'instruction CLEAR annule également l'effet de ON ERROR GOTO n°.

Exemple:

```
10 ON ERROR GOTO 100
20 INPUT X
30 PRINT 1/X
40 ON ERROR GOTO 0
50 END
100 IF ERR=11 THEN PRINT "Division impossible"
110 RESUME 20
```

ON expression GOSUB liste de n° de ligne
ON expression GOTO liste de n° de ligne
instructions

Permet le branchement à une ligne ou à un sous-programme dont la ligne est déterminée par la valeur de l'expression.

L'expression est évaluée et sa valeur, après arrondi à l'entier le plus proche, donne le rang dans la liste du sous-programme à appeler. Si par exemple cette valeur est 4, l'exécution passe à la ligne ou au sous-programme indiqué par le 4e numéro.
Si la valeur de l'expression est nulle ou supérieure au nombre de numéros de ligne de la liste, l'exécution se poursuit à l'instruction suivante.
Les numéros de lignes correspondant aux valeurs de l'expression inutilisées peuvent être omis.

Exemple:

```
10 INPUT "Numéro ";N
20 ON N GOSUB 100,,200
30 END
100 PRINT "SOUS-PROGRAMME 1"
110 RETURN
200 PRINT "SOUS-PROGRAMME 2"
210 RETURN
```

suivant que l'on répond 1 ou 3 ,
le sous-programme correspondant est exécuté

Exemple:

```
10 INPUT "Nombre ";N
20 ON SGN(N)+2 GOTO 100,200
30 PRINT 'Positif"
40 END
100 PRINT "Négatif"
110 END
200 PRINT "Nul"
210 END
```

ON INTERVAL=nombre GOSUB n° de ligne
ON INTERVAL=nombre GOTO n° de ligne
instructions

Permet de placer une interruption logicielle sur l'horloge interne.

Le nombre indique en dixièmes de seconde la valeur de l'horloge qui déclenche l'interruption.
L'exécution de INTERVAL ON lance le comptage sur l'horloge interne. L'horloge compte en dixièmes de seconde. Quand le nombre est atteint, l'exécution du programme est suspendue et la séquence d'instructions commençant au numéro indiqué est exécutée.
Si GOSUB est utilisé, la séquence doit se terminer par RETURN qui reprend l'exécution là où elle a été suspendue.
L'exécution de INTERVAL OFF arrête le comptage et par conséquent les interruptions.
L'exécution de la séquence d'instructions peut être interrompue par une autre interruption.
Si la séquence d'instructions est assez longue, il faut exécuter INTERVAL OFF au début pour ne pas être interrompu en cours d'exécution.

Exemple:

```
10 ON INTERVAL=1800 GOTO 100
20 INTERVAL ON
30 DO: LOOP
100 PRINT "C'EST CUIT"
110 END
```

Le message est affiché trois minutes exactement
après l'exécution de la ligne 20

ON KEY=caractère GOSUB n° de ligne
ON KEY=caractère GOTO n° de ligne
instructions

Permet de définir des interruptions logicielles sur la frappe d'un caractère.

Le caractère est indiqué par une constante d'un seul caractère ("A") ou par le premier caractère d'un constante ("AZERTY") ou par son code ASCII (65).
Après l'exécution de cette instruction, l'appui sur la touche correspondant au caractère provoque la suspension de l'exécution et le débranchement à la ligne de numéro indiqué.
Le retour à l'endroit de l'interruption se fait par RETURN si GOSUB est utilisé.
Il est possible de spécifier dix touches différentes avec cette instruction.
Les touches spécifiées ne peuvent plus être lues par INKEY$.
Si le numéro de ligne est égal à 0, l'interruption définie est rendue inactive pour la touche considérée.

Exemple:

```
10 ON KEY="?" GOSUB 100
20 DO: PRINT "N'importe quoi" : LOOP
30 END
100 PRINT "Voici quelques explications: ....."
110 RETURN
```

ONPEN GOSUB liste de n° de ligne
ONPEN GOTO liste de n° de ligne

instructions

Permet un branchement conditionnel (GOTO) ou un appel de sous-programme (GOSUB) dépendant de la zone visée par le crayon optique.

Les zones de l'écran doivent avoir été définies au préalable par l'instruction PEN.
L'exécution de ONPEN se déroule de la manière suivante:
— test du contact du crayon optique,
— lorsque le contact est fermé, lecture de la zone visee,
— branchement à la ligne dont le numéro a le même rang dans la liste que celui de la zone visée dans l'instruction PEN.

Le rang dans la liste est compté à partir de 0.
Si le crayon optique vise un point situé hors des zones définies, l'exécution se poursuit en séquence.

Exemple:

```
10 CLS
20 PEN 0;(10,15)-(30,35)
30 PEN 1;(40,15)-(60,35)
40 PEN 2;(70,15)-(90,35)
50 ONPEN GOSUB 100,200,300
70 END
100 PRINT "ZONE 1"
110 RETURN
200 PRINT "ZONE 2"
210 RETURN
300 PRINT "ZONE 3"
310 RETURN
```

OPEN "I", # n° de canal, descripteur de fichier.
OPEN "O", # n° de canal, descripteur de fichier
instructions

Ouvre un fichier séquentiel en lecture ou en écriture sous le n° de canal indiqué.

Toutes les opérations sur ce fichier devront se faire sous ce n° de canal. Le n° de canal doit être compris entre 1 et 16. Le périphérique pris par défaut est le lecteur 0 ou celui fixé par DEVICE.

OPEN "I" ouvre le fichier en lecture. Un même fichier peut être ouvert en lecture sous plusieurs canaux simultanément.

Exemple:

OPEN "I", #2, "1:RESUL.DAT" ouvre en lecture le fichier RESUL-.DAT sur le lecteur 1 sous le numéro 2

OPEN "O" ouvre le fichier en écriture. Un même fichier ne peut être ouvert qu'une fois en écriture. Sur disque, l'ouverture déclenche l'initialisation du fichier; si un fichier existait déjà sous le même nom, il est supprimé.

Exemple:

OPEN "0",#1, "2: TELEPH.DAT" ouvre en écriture le fichier TELEPH.DAT sur le lecteur 2 sous le numéro 1
OPEN "0",#3, "LPRT:" ouvre en écriture le fichier numéro 3 sur l'imprimante parallèle

OPEN "D", # n° de canal, descripteur de fichier , longueur
OPEN "R", # n° de canal, descripteur de fichier , longueur
Instructions

Ouvre le fichier à accès direct dont le nom est indiqué, en lecture et en écriture, sous le n° de canal précisé.

Le n° de canal doit être compris entre 1 et 16.
La longueur, optionnelle, fixe la longueur de l'enregistrement. Cette longueur doit être inférieure ou égale à la longueur indiquée dans FILES. Si la longueur n'est pas précisée dans OPEN, l'enregistrement occupe alors un secteur, soit 128 octets en simple densité ou 255 octets en double densité.
Le périphérique pris par défaut est le lecteur 0 ou celui fixé par DEVICE.
On ne peut ouvrir de fichier à accès direct que sur disquette. Les deux formes OPEN "D", et OPEN "R",... sont équivalentes.

Exemple:

OPEN "D", #2, "1:AGENDA.OLD",32 ouvre le fichier à accès direct AGENDA.OLD sous le numéro 2, avec une longueur d'enregistrement de 32 caractères

PAINT (colonne, ligne) , couleur
instruction

Remplit avec le motif en cours une zone de couleur homogène en partant du point situé en (colonne, ligne) avec la couleur précisée.

Si la couleur n'est pas précisée, c'est la couleur courante qui est prise par défaut.

Le remplissage se fait avec le motif défini par l'instruction PATTERN. Par défaut, le remplissage est uniforme.
Dans tous les cas, le remplissage ne modifie qu'une seule couleur, forme ou fond suivant le signe.

Exemple:

PAINT (100,125), 3 remplissage à partir du point 100,125 par la couleur jaune

PATTERN caractère

instruction

Fixe le motif de remplissage.

Le caractère qui servira de motif peut être donné sous forme de chaîne de caractères (c'est le premier caractère de la chaîne qui est pris) ou par son code ASCII. Si le code ASCII est supérieur à 127, il s'agit alors d'un caractère graphique défini par l'utilisateur.
Le motif est utilisé par la suite dans toutes les instructions de remplissage: BOXF, CIRCLEF, PAINT.

Exemple:

PATTERN "/" le caractère / servira de motif
PATTERN 129 le 2e caractère graphique servira de motif
PATTERN GR$(1) ici également

PEEK(adresse)

fonction

Retourne le contenu de l'octet situé à l'adresse indiquée.

L'adresse est un nombre compris entre −65536 et 65535.
Le résultat est un entier compris ente 0 et 255.
En Basic 128, si l'adresse est comprise entre &HA000 et &HDFFF, l'octet est celui de la banque courante, c'est-à-dire celle fixée par BANK ou, par défaut, la banque la plus élevée.

Exemple:

BANK 1: PRINT PEEK(&HA100) donne le contenu de l'octet situé en &HA100 dans la banque 1

PEN liste de zones
avec zone =
numéro;(colonne1,ligne1)–(colonne2,ligne2)
instruction

Définit des zones rectangulaires sur l'écran et leur attribue un numéro pour l'utilisation de ONPEN.

Les numéros de zones sont compris entre 0 et 7.
Les zones peuvent être définies en mode caractère ou en mode graphique.

- mode caractère:

Chaque zone correspond à un seul caractère; elle est définie par un seul point dont les coordonnées sont données en caractères: colonnes de 0 à 39 et lignes de 0 à 24.

Exemple:

`PEN 3;(30,15),4;(32,15)` définit les zones 3 et 4 en mode caractère

- mode graphique:

Chaque zone est définie par les deux sommets opposés du rectangle comme dans l'instruction BOX. Les coordonnées sont exprimées en graphique: colonnes de 0 à 319 et ligne de 0 à 199.

Exemple:

`PEN 0;(10,10)–(40.40).1;(50.10)–(80,40)` définit deux zones numérotées 0 et 1

Si un numéro de zone est défini sans être suivi des coordonnées, la zone est supprimée et ne peut plus être sélectionnée par ONPEN.
Une instruction PEN sans argument supprime toutes les zones définies.
Si deux zones ont une partie commune, lorsque le crayon optique vise cette partie commune, ONPEN effectue le branchement correspondant à la zone de plus petit numéro.

PLAY liste de chaîne de caractères
Instruction

Joue la musique définie dans les chaînes de caractères.

Les chaînes de caractères ne peuvent comporter que des notes, des silences ou des attributs.
— notes: DO RE MI FA SO LA SI suivies de # ou b si l'on veut un dièse ou un bémol,
— silence: P
— attributs: ce sont l'attaque, la longueur, l'octave et le tempo.

Signification	Attribut	Domaine	Valeur par défaut
amortissement	A	0-255	0 note continue
longueur	L	1-96	24 noire
octave	O	1-5	4
tempo	T	1-255	5

Amortissement:
A0 correspond à un son continu, A225 à un son très amorti.
longueur:
L96 correspond à une ronde, L48 à une blanche,... , L3 à une triple croche .
Octave:
O1 est l'octave la plus grave, O5 la plus aiguë.
tempo:
T1 est le tempo le plus rapide, T255 le plus lent.
Un attribut A, L, O, T agit sur toutes les notes qui suivent jusqu'à ce qu'un nouvel attribut vienne modifier sa valeur.
Les espaces sont autorisés à n'importe quel endroit dans la chaîne; le point-virgule peut servir à séparer les notes.

Exemple:

```
10 A$="O5 DOREMIDOREPREMIFAFAMIPDOREMIDOREPREMIFASODO"
20 PLAY A$,A$
```

POINT (colonne,ligne)
fonction

Donne le numéro de la couleur du point indiqué.

Les coordonnées sont indiquées en graphique: colonne de 0 à 319 et ligne de 0 à 199.
Le numéro est positif si le point appartient à la forme, négatif s'il appartient au fond.

POKE adresse, nombre
instruction

Dépose le nombre dans l'octet situé à l'adresse indiquée.

Le nombre doit être compris entre 0 et 255.
L'adresse doit être comprise entre −65536 et 65535.
En Basic 128, si l'adresse est comprise entre &HA000 et &HDFFF, l'octet est celui de la banque courante c'est-à-dire celle fixée par BANK ou, par défaut, la banque la plus élevée.
Cette instruction est interdite sur la mémoire morte.

POS (n° de canal)
fonction

Donne la position du pointeur sur la ligne.

La position la plus à gauche est notée 0.
Le numéro de canal indique le périphérique concerné.
Si ce numéro est égal à 0 ou s'il est omis, la fonction renvoie la position du curseur sur l'écran (nombre compris entre 0 et 39).
La fonction CSRLIN permet de connaître le numéro de la ligne où se trouve le curseur.

PRINT liste de données
instruction

Affiche sur l'écran les données dans l'ordre de la liste.

Le mot clé PRINT peut être remplacé par un point d'interrogation.
L'affichage commence à l'endroit où se trouve le curseur.
Un point-virgule permet l'affichage de la donnée suivante juste après la précédente.
Une virgule permet l'affichage de la donnée suivante au début de la tabulation suivante. Les tabulations fixes sont espacées de treize caractères à partir du caractère 0.
Si la liste de données ne se termine pas par une virgule ou un point-virgule, l'affichage se termine par le passage à la ligne suivante.
Si les données occupent plus de 40 caractères, l'affichage se poursuit sur la ligne suivante.

Si une donnée doit être affichée en fin de ligne et qu'il ne reste pas assez de place, l'affichage se fait automatiquement sur la ligne suivante. Dans ce cas, les minuscules accentuées sont comptées comme trois caractères.
Les nombres sont toujours suivis d'un espace. Les nombres positifs sont précédés d'un espace, les nombres négatifs du signe moins.
L'affichage se fait avec exposant si la valeur du nombre est en dehors de l'intervalle 1E−6,1E+6 pour la simple précision, ou 1D-16 1D+16 pour la double précision.
Les chaînes de caractères sont affichées telles quelles.
Avec des caractères en double hauteur, le retour à la ligne provoque un double interligne. Cependant, le premier affichage se fait sur la ligne du curseur et sur la ligne supérieure. Il est donc souvent nécessaire de le faire précéder d'un PRINT vide.

Exemple:

```
10 X=1
20 PRINT X,X+1,X+2
30 PRINT X;X+1;X+2
40 PRINT "X=";X
```

PRINT # n° de canal , liste de données
instruction

Ecrit les données (variables ou constantes) dans le fichier dont on précise le n° de canal.
Les données sont enregistrées de la même manière qu'à l'écran avec PRINT (espaces, séparateurs,...).
Le fichier peut être séquentiel ou à accès direct. Dans ce dernier cas, PRINT # écrit dans le tampon d'enregistrement. Le transfert de l'enregistrement dans le fichier doit se faire par PUT #.

Exemple:

```
PRINT #2, A$, N
PRINT #1, "ARTHUR
```

PRINT USING chaîne de caractères; liste de données
instruction

Permet des affichages selon un format décrit dans la chaîne de caractères.

Les données de la liste sont séparées par des virgules ou des points-virgules.
La chaîne de caractères contient des caractères qui précisent le format d'affichage. Leur signification est la suivante:

format des chaînes de caractères

! signifie que seul le premier caractère de la chaîne doit être affiché.
% % signifie que la longueur exacte de la zone d'affichage de la chaîne est la même que celle comprise entre les deux % compris. La chaîne est cadrée à gauche à l'affichage, les caractères en trop étant éliminés ou l'espace restant étant complété par des espaces.
& signifie que la chaîne doit être affichée telle quelle. Ceci permet d'éviter des sauts de ligne pour une chaîne trop longue.

Exemple:

```
10 A$="QU'IMPORTE LE FLACON"
20 PRINT USING "!";A$
30 PRINT USING "% %";A$
40 PRINT USING "&";A$
RUN
Q
QU'IMPORTE
QU'IMPORTE LE FLACON
```

format des nombres

représente la position de chaque chiffre. L'affichage d'un nombre est cadré à droite. Si la zone est trop petite le dépassement est indiqué par l'affichage du caractère % en tête du nombre. Si la zone est trop grande, le reste est complété par des espaces.
. représente le point décimal. Il peut être placé n'importe où pour séparer la partie entière de la partie décimale. Si nécessaire, les nombres sont arrondis avant affichage.

Exemple:

```
10 A$=###.##
20 PRINT USING A$;12
30 PRINT USING A$;−12.345
40 PRINT USING A$;.1234
50 PRINT USING A$;1234.5678"
```

```
RUN
 12
-12.35
 0.12
%1234.57
```

+ placé en début ou en fin de format signifie que le signe + doit toujours précéder ou suivre un nombre positif.

– placé en fin de format signifie que le signe – doit toujours suivre un nombre négatif.

Exemple:

```
PRINT USING "+##.#";12.34
 +12.3
PRINT USING "##.#-";-12.34
 12.3-
```

** placé en tête de format signifie que les espaces en tête doivent être remplacés par des astérisques.

Exemple:

```
PRINT USING "**##.#";1.23
**   .2
```

$$ placé en tête de format signifie que le signe dollar doit être affiché immédiatement à gauche du nombre.

Exemple:

```
PRINT USING "$$##.#";1.23
 $1.2
```

**$ placé en tête de format combine les deux effets précédents.

Exemple:

```
PRINT USING "**$##.#";1.23
***$1.2
```

,situé à gauche du point décimal signifie que les puissances de 1000 doivent être séparées par une virgule dans la partie entière.

Exemple :

```
PRINT USING "#######,.##";1234567
 1,234,570.00
```

^^^^ placés en fin de format signifie que la notation exponentielle doit être utilisée.

Exemple:

```
PRINT USING "##.##^^^^";123.45
 1.23E+02
```

format général

Le format doit contenir autant de formats élémentaires qu'il y a de données à afficher. Les formats élémentaires peuvent être séparés par des constantes qui seront affichées entre les données.
Si le format général ne contient pas assez de formats élémentaires, il est repris au début pour l'affichage de la suite des données.

Exemple:

```
PRINT USING "J'AI VENDU ### LOGICIELS A ###.## FRANCS";53,499.50
J'AI VENDU 53 LOGICIELS A 499.50 FRANCS
PRINT USING "##.#";1.23,2.34;3.45
 1.2 2.3 3.5
```

PRINT # n° de canal, USING chaîne de caractères; liste de données

instruction

Fonctionne comme PRINT #, mais les données sont écrites suivant le format précisé par USING.
Les données sont enregistrées de la même manière qu'à l'écran avec PRINT USING.

Exemple:

```
PRINT #1, USING "###.##"; 120.5
```

PSET (colonne,ligne) caractère,couleur,fond ,inversion
PSET (colonne,ligne) ,couleur
instruction

Marque un point situé en colonne,ligne.
Cette instruction fonctionne en mode caractère quand l'argument caractère est présent et en mode graphique quand il est absent.
En mode caractère, la colonne est un nombre de 0 à 39 et la ligne un nombre de 0 à 24.
En mode graphique, colonne et ligne sont des nombres compris entre −32768 à 32767 en Basic 128.

La suite de la syntaxe est identique à celle de BOX.

Exemple:

PSET (10,20) "X",3,0 marque le caractère X en colonne 10 ligne 20, de couleur jaune sur fond noir
PSET (55,115) marque le point graphique situé en 55,115 avec la couleur courante

PTRIG
fonction

Rend la valeur VRAI (−1) si le contact du crayon optique est fermé et la valeur FAUX (0) sinon.

PUT (colonne , ligne) ,élément de tableau
instruction

Restitue à l'écran la portion d'image conservée dans le tableau à partir de l'élément indiqué.

La portion d'image est obligatoirement composée d'un nombre entier de caractères. Les coordonnées du sommet sont donc données en caractères: colonne de 0 à 39 et ligne de 0 à 24.
Le sommet indique le coin supérieur gauche de l'image.
L'image doit avoir été mémorisée dans le tableau numérique entier

par une instruction GET préalable, ou chargée dans le tableau par une instruction LOADP.
La portion d'image n'est pas affichée si elle est trop grande.

Exemple:

PUT (10,12),IM%(300) restitue la portion d'image conservée dans le tableau à partir de l'élément 299 , au point situé en colonne 10 ligne 12

PUT # n° de canal, n° d'enregistrement

instruction

Transfère un enregistrement d'un fichier à accès direct (dont on donne le n° de canal) de la zone tampon vers la disquette.
Le contenu de l'enregistrement doit avoir été déposé par WRITE# ou PRINT#, ou directement dans les variables de champ par LSET, RSET ou MID$()=. Si le numéro d'enregistrement n'est pas précisé, PUT # écrit dans l'enregistrement suivant du fichier, ou dans le 1er si aucun enregistrement n'a été lu ou écrit.

Exemple:

PUT #1,12 dépose le douzième enregistrement du fichier n° 1

READ liste de variables

instruction

Lit les données des instructions DATA et les affecte aux variables de la liste.
Les variables doivent être de même type que les données à lire, sinon une erreur "Syntax Error" est détectée.
Une seule instruction READ peut lire plusieurs lignes de DATA et inversement, une ligne de DATA peut être lue par plusieurs instructions READ.
La première instruction READ commence par lire les données de la première instruction DATA du programme. La suite des lectures se fait en séquence.
S'il n'y a plus assez de données, une erreur "Out of Data" apparaît.
Il est possible de commencer ou de recommencer la lecture à une ligne donnée par une instruction RESTORE.

Exemple:

```
10 READ NOM$,PR$,AGE
20 PRINT NOM$,PR$,AGE
100 DATA MARTIN, LUCIEN, 77
```

REM texte
instruction

Permet d'incorporer le texte de remarques, titres, commentaires,... dans un programme.
Le mot clé REM peut être remplacé par une apostrophe.
Tout ce qui suit REM jusqu'à la fin de la ligne est ignoré par l'interpréteur mais apparaît dans le listage du programme.
Il est possible de brancher à une ligne contenant un REM, l'exécution se fait à la première ligne qui suit.
Des commentaires peuvent être ajoutés en fin de ligne en les séparant des instructions par une apostrophe.

Exemple :

```
10 REM Titre du programme 1 jan 1984
20 PRINT "Bonjour" ' message de bienvenue
30 A=2+2 ' calcul
40 PRINT "Bonsoir" : REM message de politesse
```

RENUM nouveau n°, 1re ligne ancien n°, 1re ligne ancien n°, dernière ligne pas

Rénumérote un programme ou une partie de programme.

Toutes les indications sont optionnelles et possèdent des valeurs par défaut.
Le premier nombre indique le nouveau numéro (10 est pris par défaut).
Le deuxième nombre indique à partir de quel numéro (ancien) il faut commencer à renuméroter (par défaut, c'est la première ligne du programme).
Le troisième nombre indique l'ancien numéro de ligne de fin de la renumération (par défaut, la fin du programme).
Le quatrième nombre indique l'écart entre chaque nouvelle ligne (par défaut 10).

RENUM change en conséquence, et dans tout le programme, tous les numéros des lignes renumérotées dans les branchements et les appels de sous-programmes. Si dans la ligne L1 (ancien numéro), il est fait appel à une ligne L2 inexistante, le message suivant apparaît:
Undefined line L2 in L1
dans lequel L1 est le numéro de ligne renumérotée.
Les virgules sont indispensables, sauf les dernières.
La renumérotation d'une partie de programme n'est possible que si les nouvelles lignes restent à l'intérieur de l'espace laissé par les lignes inchangées: le nouveau n° de la 1re ligne renumérotée doit rester plus grand que le n° de la ligne inchangée qui le précède, et, le nouveau n° de la dernière ligne renumérotée doit rester plus petit que le n° de la ligne inchangée qui suit.

Exemple:

RENUM 1500, 1000, 1340, 5 renumérote la partie de programme comprise entre 1000 et 1340, avec un pas de 5, en commençant à 1500

RENUM 100, 10 renumérote le programme à partir de 10 avec un pas de 10, en commençant à 100

RENUM renumérote tout le programme avec un pas de 10 en commençant à 10

RESET
instruction

Initialise le micro-ordinateur.

Son effet est de ramener au menu initial.

Exemple:

RESET

RESTORE n° de ligne
instruction

Permet de relire des constantes dans une instruction DATA d'une ligne donnée.

Après l'instruction RESTORE, le premier READ prend ses valeurs dans le premier DATA rencontré à partir de cette ligne.
Si aucun numéro de ligne n'est indiqué, le premier READ prend ses valeurs à partir du premier DATA du programme.

Exemple:

```
10 READ A,B
20 PRINT A,B
30 RESTORE
40 READ A
50 PRINT A
100 DATA 12,23,34
```

RESUME n° de ligne
RESUME NEXT
instruction

Permet le retour au programme principal après une séquence de traitement d'erreur appelée par ON ERROR GOTO.

L'exécution reprend au numéro de ligne indiqué s'il est présent ou à l'instruction qui a provoqué l'erreur si rien n'est indiqué.
Si l'option NEXT est présente, l'exécution reprend à l'instruction qui suit celle qui a provoqué l'erreur.
Si cette instruction est rencontrée sans appel par ON ERROR GOTO, une erreur "Resume without Error" est détectée.

Exemple:

```
10 ON ERROR GOTO 100
20 INPUT X
30 PRINT SQR(X)
40 ON ERROR GOTO 0
50 END
100 IF ERR=5 THEN "Nombre negatif"
110 RESUME 20
```

RETURN
instruction

Retour au programme principal après un appel par GOSUB.

(Voir instruction GOSUB.)

RIGHT$(chaîne de caractères, longueur)
fonction

Donne la partie droite de la chaîne sur une longueur précisée.

Si la longueur est nulle, le résultat est la chaîne vide.
Si la longueur demandée est supérieure à la longueur de la chaîne, le résultat est la chaîne entière.

Exemple:

```
? RIGHT$("MELODRAME",5)
DRAME
```

RND(nombre)
fonction

Donne un nombre réel, pseudo-aléatoire, compris entre 0 et 1 exclus.

Les nombres obtenus avec cette fonction sont différents et répartis uniformément dans l'intervalle ouvert 0,1 si l'argument de la fonctior est un nombre positif ou s'il n'y a pas d'argument.
La séquence des nombres générés est réinitialisée à chaque exécution du programme.
Un argument négatif réinitialise la séquence à partir d'une valeur qui dépend de l'argument.

Exemple:

```
10 DO
20   FOR I=1 TO 5
30   PRINT RND
40   NEXT I
50   PRINT RND(-2)
60 LOOP
RUN
```

ROT
fonction

Rend l'orientation de la tortue active.

Le résultat est un nombre compris entre 0 et 255. Le sens positif est celui des aiguilles d'une montre.
La tortue active est la dernière fixée par TURTLE.

Exemple:

```
? ROT
167
```

ROT TO nombre
instruction

Fixe ou modifie l'orientation de la tortue active.

Si l'option TO est présente, l'orientation est fixée de façon absolue, sinon elle est modifiée relativement à sa valeur antérieure.
La valeur absolue du nombre est comprise entre 0 et 255. Si cette valeur est plus grande, seul le reste de sa division par 256 (modulo) est pris en compte.
Un nombre positif signifie vers la droite, un nombre négatif signifie vers la gauche.

La tortue active est la dernière fixée par TURTLE.

Exemple:

ROT TO −32	fixe l'orientation à −45° vers la gauche
ROT 64	modifie l'orientation de 90° vers la droite
ROT 320	en fait autant

RSET variable chaîne = chaîne de caractères
instruction

Dépose une chaîne de caractères dans une variable chaîne en la cadrant à droite.

La chaîne de caractères est déposée en commençant par la droite. Si sa longueur est plus petite que la longueur de la variable , le reste

est complété à gauche par des espaces; si sa longueur est trop grande, la fin de la chaîne est éliminée. Les nombres peuvent être déposés dans une variable chaîne après avoir été convertis par:

MKI$( ) pour un nombre entier
MKS$( ) pour un nombre en simple precision
MKD$( ) pour un nombre en double précision.

Cette instruction est utilisée le plus souvent pour déposer des données dans une variable de champ définie dans FIELD.

Exemple:

`RSET ID$ = NOM$+PR$` dépose dans la variable ID$ le contenu des deux chaînes NOM$ et PR$ en cadrant le tout à droite
`RSET Q$ = MKI$(1234)` dépose dans Q$ le nombre 1234 après conversion

RUN n° de ligne
RUN descripteur de fichier , R
commande

Lance l'exécution, après chargement éventuel, du programme résident en mémoire.

Si un n° de ligne est indiqué, l'exécution du programme résident commence à la ligne donnée.
Si un fichier est indiqué, le programme qu'il contient est chargé en mémoire puis exécuté à partir du début.
Avec l'option R, les fichiers déjà ouverts avant le lancement restent ouverts.

Exemple:

`RUN 100` lance l'exécution en ligne 100
`RUN "1:OTHELLO", R` charge le programme OTHELLO et l'exécute à partir du début tout en conservant tous les fichiers ouverts.

SAVE descripteur de fichier
SAVE descripteur de fichier, A
SAVE descripteur de fichier, P
instruction

Sauvegarde, sous le nom et sur le support indiqués, le programme présent en mémoire centrale.

Sur disque, si un fichier existait déjà sous le même nom, il est remplacé par le nouveau programme.
Le suffixe par défaut est BAS.
Avec l'option A, le programme est sauvegardé sous une forme littérale, identique au listage à l'écran. Il pourra alors être utilisé par MERGE.

Avec l'option P (protégé), le programme est sauvegardé sous une forme codée. Au chargement suivant, le programme ne pourra être ni listé ni modifié. Il est conseillé de le sauvegarder également sous la forme habituelle.

Exemple:

`SAVE "CASS:OTHELLO", P` sauvegarde sur cassette le programme résident, sous le nom OTHELLO.BAS, sous forme protégée
`SAVE "TICTAC", A` sauvegarde sous forme littérale (ASCII)
`SAVE "TOE"` sauvegarde normale

SAVEM descripteur de fichier, adresse 1, adresse 2, adresse 3

instruction ou commande

Sauvegarde une partie de la mémoire dans un fichier binaire.

L'adresse 1 indique le début de la zone à sauvegarder, adresse 2 en indique la fin, adresse 3 indique l'adresse de lancement de l'exécution au moment du chargement par LOADM avec l'option R ou par EXEC sans paramètre.
Si ses adresses sont comprises entre &HA000 et &HDFFF, la zone mémoire est prise dans la banque de mémoire courante.
SAVEM peut servir à conserver une zone mémoire précise ou un programme en binaire.

Exemple:

`SAVEM "HORLOGE", &H 8800, &H 8900, &H 8800` sauvegarde la zone binaire comprise entre &H8800 et &H8900

SAVEP descripteur de fichier, élément de tableau
instruction

Sauvegarde dans un fichier une portion d'écran compactée dans un tableau.

Le tableau doit être un tableau d'entiers. Il peut contenir plusieurs images mais une seule est sauvegardée à la fois.
L'image doit avoir été compactée à partir de l'élément indiqué par une instruction GET préalable.
Le suffixe par défaut est MAP.
L'image peut être récupérée ultérieurement par LOADP et affichée par PUT.

Exemple:

`SAVEP "EINSTEIN", IM%(300)` sauvegarde l'image compactée à partir de l'élément 299 dans le tableau IM% sous le nom EINSTEIN.MAP

SCREEN(colonne.ligne)
fonction

Donne le code ASCII du caractère affiché en colonne et ligne indiqués.

Les coordonnées sont données en caractères: colonne de 0 à 39, ligne de 0 à 24.
Seuls les caractères du tableau ASCII et les minuscules accentuées sont reconnus. Si aucun caractère n'est reconnu, le résultat est 0.
Les codes suivants sont attribués aux minuscules accentuées et au ç:

128	ç	134	ê	140	é
129	á	135	ë	141	û
130	â	136	è	142	ü
131	ä	137	î	143	ù
132	à	138	ï		
133	é	139	ô		

Exemple:

```
10 LOCATE 2,4: PRINT "A"
20 PRINT SCREEN(2,4)
RUN
 A
 65
```

SCREEN forme, fond, tour, inversion, incrustation

instruction

Fixe les couleurs de l'affichage pour tout l'écran.
Tous les paramètres sont facultatifs mais un au moins doit être présent.

Les trois premiers paramètres indiquent respectivement:
— la couleur des caractères et des tracés,
— la couleur du fond,
— la couleur du pourtour de l'écran.
La présence du quatrième paramètre indique que les couleurs de forme et de fond doivent être inversées.
La présence du cinquième paramètre indique que l'image TV doit être incrustée. Cette option ne fonctionne qu'avec l'interface d'incrustation; elle provoque une erreur fatale en l'absence d'interface.

Exemple:

```
SCREEN 1,0,5     caractères rouges sur fond noir et tour magenta
SCREEN ,,,1      inverse les couleurs précédentes
```

SCREENPRINT

instruction

P.270
Recopie le contenu de l'écran sur l'imprimante parallèle.
cette instruction ne fonctionne que pour les imprimantes PR 90-040, PR 90-042, PR 90-582 et PR 90-600.

Exemple

```
SCREENPRINT
```

SEARCH chaîne de caractères, n° ligne 1 - n° ligne 2
commande

Affiche sur l'écran toutes les lignes du programme comportant la chaîne de caractères spécifiée, de la ligne 1 à la ligne 2 incluse.

La chaîne de caractères à rechercher peut être donnée dans une constante ou une variable.
On peut effectuer la recherche sur tout ou partie du programme, ou sur la ligne courante, avec une syntaxe identique à celle de LIST.

Exemple:

`SEARCH "NOM$", 1000-2000` recherche la variable NOM$ entre les lignes 1000 et 2000
`SEARCH "NBJ", .` recherche NBJ sur la ligne courante
`SEARCH "520"` recherche 520 dans tout le programme

SGN(nombre)
fonction

Donne le signe du nombre.

Le résultat vaut:
— 1 si le nombre est positif,
— 0 si le nombre est nul,
— −1 si le nombre est négatif.

Exemple:

```
PRINT SGN(-2)
-1
```

SHOW
fonction

Indique si la tortue active est visible (résultat −1) ou invisible (résultat 0).

Exemple:

```
? SHOW
-1
```

SHOW visible, libre
instruction

Fixe l'état de la tortue active.

Le premier paramètre fixe la visibilité.
S'il vaut 0, la tortue est invisible; s'il vaut 1, la tortue est visible.
Le deuxième paramètre fixe la liberté.
S'il vaut 1, la tortue est affichée à chaque modification.
S'il vaut 0, la tortue n'est affichée qu'à chaque déplacement.

Exemple:

`SHOW 1,1` la tortue active est visible et libre

SIN(nombre)
fonction

Donne le sinus du nombre compté en radians.

En Basic 128, le résultat est en double précision si l'argument est en double précision.

Exemple:

```
PRINT SIN (1.5)
.997495
```

SKIPF "nom de fichier"
instruction

Sur la cassette, arrête la bande après le fichier indiqué, ou le fichier suivant si rien n'est indiqué.

Exemple:

`SKIPF "TICTAC.BAS"` arrête la bande après le fichier TICTAC

SPACE$ (nombre)
fonction

Donne une chaîne de caractères composée d'autant d'espaces que le nombre le précise.

Exemple:

`SPACE$ (30)` donne une chaîne de 30 espaces

SPC(nombre)
fonction

Cette fonction n'est utilisable que dans un PRINT.

Exemple:

```
PRINT "A" SPC(10) "B"
A          B
```

SQR(nombre)
fonction

Donne la racine carrée du nombre.

En Basic 128, le résultat est en double précision si l'argument est en double précision.
Si l'argument est négatif, l'erreur "Illegal Function Call" est détectée.

Exemple:

```
PRINT SQR(10)
 3.16228
```

STICK(n° de manette)
fonction

Donne la position de la manette de jeu désignée.

Il y a deux manettes de jeu numérotées 0 et 1.
Le résultat vaut:
0 neutre
1.90°
2.45°
3.0°
4.−45°
5.−90°
6.−135°
7.180°
8.135°

Exemple:

ON STICK(0) GOSUB ... branche au sous-programme correspondant à la position de la manette

STOP
instruction

Suspend l'exécution du programme.

Cette instruction peut être placée en n'importe quel endroit du programme.
Lorsqu'elle est rencontrée, le message "Break in ..." apparaît.
La reprise peut être faite par CONT ou GOTO.
Tous les fichiers restent ouverts.

Exemple:

```
10 INPUT A,B
20 C=A^2+B^2
30 STOP
40 PRINT C
```

STR$(nombre)
fonction

Transforme un nombre en sa représentation en chaîne de caractères.

Le résultat a la même forme que l'affichage du nombre par PRINT.
La fonction inverse de STR$ est VAL.

Exemple:

```
PRINT STR$(68.5);STR$(-1234)
 68.5 -1234
```

STRIG(n° de manette)
fonction

Donne l'état du bouton de la manette désignée.
Il y a deux manettes de jeu numérotées 0 et 1.
Le résultat est −1 si le bouton est enfoncé, 0 sinon.

Exemple:

```
IF STRIG(1) THEN PRINT "TOUCHE"
```

STRING$ (nombre, caractère)
fonction

Donne une chaîne de caractères identiques au caractère indiqué et dont la longueur est égale au nombre indiqué.

Le caractère peut être précisé par son code ASCII.

Exemple:

`STRING$ (10," * ")`	donne une chaîne de 10 étoiles
`STRING$ (20,42)`	donne une chaîne de 20 étoiles

SWAP variable 1, variable 2
instruction

Echange le contenu de deux variables de même type.

Cette instruction est très utile dans les programmes de tris.

Exemple:

`SWAP A(I),A(J)`	échange les valeurs des éléments I et J du tableau A

TAB(nombre)
fonction

Positionne le curseur dans une instruction PRINT ou PRINT#.
Le nombre indiqué, modulo la largeur du périphérique utilisé (écran ou imprimante), indique la position du curseur en colonne.
Si cette position est à gauche de la position courante, un saut de ligne est effectué.
Un nombre négatif ou nul est sans effet.
En double largeur, le positionnement tient compte de la largeur double.
Les règles de positionnement fixées par les séparateurs virgule et point-virgule sont respectées.

Exemple:

```
PRINT "A";TAB(10);"B"
A        B
```

TAN(nombre)
fonction

Donne la tangente du nombre exprimé en radians.

Le résultat est en double précision si l'argument est en double précision.

Exemple:

```
? TAN(1.5)
 14.1014
```

TRACE
fonction

Indique si la tortue active laisse une trace de son déplacement (résultat −1) ou non (résultat 0).

Exemple:

```
? TRACE
-1
```

TRACE nombre
instruction

Fixe ou enlève la trace de la tortue active.

Si l'argument vaut 0 (FAUX), la tortue ne trace pas, sinon elle trace.

Exemple:

TRACE 1 la tortue laissera une trace de tous ses déplacements

TROFF
instruction

Annule le mode de trace du programme déclenché par TRON et ferme le fichier de trace s'il y a lieu.

TRON descripteur de fichier , n° ligne 1 - n° ligne 2
instruction

Déclenche le mode de trace du programme.

Le mode de trace provoque la visualisation sur l'écran ou l'enregistrement dans le fichier indiqué des numéros de lignes exécutés.

Les numéros tracés apparaissent entre crochets.
Si le descripteur de fichier est présent, la trace est écrite sur ce fichier, sinon elle est visualisée sur l'écran.
Seules les lignes exécutées dont le numéro est compris entre les deux numéros indiqués sont tracées. La plage des numéros de lignes est indiquée avec la même syntaxe que LIST.

Exemple:

TRON "LPRT:", 200-400 trace sur imprimante la partie de programme comprise entre les lignes 200 et 400
TRON trace tout le programme à l'écran

TURTLE n°, colonne, ligne, chaîne de caractères
instruction

Définit et place la tortue active.

Le premier paramètre indique le numéro de la tortue active. On peut définir jusqu'à dix tortues différentes, numérotées de 0 à 9.
Les deux paramètres suivants fixent la position de la tortue dans l'écran: colonne et ligne compris entre −32768 et 32767. Ces deux paramètres ne sont pas obligatoires. Quand ils sont absents, la position de la tortue n'est pas modifiée ou, si cette tortue est active pour la première fois, elle est placée au centre de l'écran.
Le quatrième paramètre, facultatif, est une chaîne de caractères qui permet de définir la forme de la tortue.
Cette chaîne est constituée de doublets indiquant chacun une rotation suivie d'un déplacement.
Il existe quatre types de doublets possibles: RaDn, RaUn, LaDn et LaUn, où a indique l'angle dont il faut tourner et n le nombre de pas dont il faut avancer.
Les angles et les nombres de pas sont des nombres compris entre 0 et 255. Un angle de 256 est égal à 360°.
R signifie rotation à droite et L rotation à gauche.
D signifie déplacement avec tracé et U déplacement sans tracé.
Les espaces ne sont pas pris en compte.

Exemple:

TURTLE 1,80,120,"L0D40L64D40" définit la tortue n° 1 en forme d'angle droit, et la place en colonne 80 ligne 120
A$="L0D40L128U20L64D20" : TURTLE 2,,,A$ définit la tortue n° 2 en forme de T et la place au centre de l'écran
TURTLE 0 active la tortue n° 0

UNLOAD n° de lecteur
instruction

Ferme tous les fichiers ouverts sur la disquette située dans le lecteur dont on précise le numéro et recopie toutes les informations utiles sur la disquette.

Par défaut, UNLOAD considère le lecteur n° 0 ou celui défini dans DEVICE.

Cette instruction garantit le retrait de la disquette du lecteur sans aucun risque, en particulier après l'arrêt d'un programme d'écriture de fichiers par CNT-C ou par une erreur.

Exemple:

`UNLOAD 1` ferme tout sur la disquette 1

UNMASK

instruction

Permet de démasquer dans la fenêtre de travail toutes les zones masquées au préalable par ATTRB.

Tous les caractères masqués, couleur noire sur fond noir, sont remis dans la couleur et le fond en vigueur au moment du masquage. UNMASK ne fonctionne que si le masquage est actif (ATTRB,,1). Le démasquage n'a d'effet que dans la fenêtre de travail courante. Le changement de fenêtre de travail par CONSOLE permet des démasquages sélectifs.

Exemple:

`CONSOLE 10,15: UNMASK` démasque toutes les zones masquées entre la ligne 10 et la ligne 15

USRn°(donnée)

fonction

Appel de la fonction utilisateur en langage machine de numéro indiqué (0 par défaut).

La fonction doit avoir été définie au préalable par DEFUSRn°. Pour utiliser cette fonction, voir Annexe 4

Exemple:

`A=USR2(F)` appel de la fonction utilisateur n° 2 avec la variable F comme argument

VAL (chaîne)
fonction

Donne la valeur numérique d'une chaîne de caractères représentant un nombre.

Si le premier caractère n'est pas un caractère décimal, ou +, ou – ou, un point ou un espace, le résultat de la fonction est nul.
La fonction inverse de VAL est STR$.

Exemple:

```
PRINT VAL ( "15 Rue Victor Hugo")
 15
```

VARPTR (variable)
fonction

Retourne l'adresse du premier octet de la variable indiquée.

Le résultat est un nombre entier compris entre –32768 et 32767. Il faut lui ajouter 65536 pour avoir l'adresse réelle quand il est négatif.

En Basic 128, après l'exécution de cette fonction, la banque courante est devenue celle où se trouve la variable cherchée. On peut obtenir ce numéro de banque par la fonction BANK.
Cette fonction est utilisée le plus souvent pour des modules en langage machine.
Pour les informations concernant l'organisation des variables en mémoire, voir Annexe 3.

Exemple:

`PRINT VARPTR(A)` donne l'adresse de la variable A

VERIFYON
VERIFYOFF
instructions

Avec l'option ON, toute écriture sur disquette sera suivie d'une vérification. L'option OFF supprime cette vérification.

Exemple:

```
VERIFY ON
```

WAIT adresse, masque 1, masque 2
instruction

Suspend l'exécution du programme jusqu'à ce qu'une configuration de bits apparaisse à l'adresse indiquée.

L'adresse est un nombre compris entre −65536 et 65535.
Les deux masques sont des nombres compris entre 0 et 255.
Le fonctionnement de l'instruction est le suivant:
— l'expression booléenne suivante est calculée:
(contenu de l'adresse XOR masque 2) AND masque 1
— l'exécution ne se poursuit que lorsque cette expression n'est pas nulle.
Le masque 2 est facultatif. Dans ce cas l'opérateur XOR n'est pas appliqué.
Cette instruction est utilisée le plus souvent pour attendre un événement sur un circuit périphérique.
On ne peut sortir de l'attente par CNT-C.
L'utilisation de cette instruction nécessite une très bonne connaissance de la structure interne du micro-ordinateur.

WINDOW (colonne 1. ligne 1) - (colonne 2. ligne 2)

instruction

Définit la fenêtre de visualisation des instructions graphiques.

La fenêtre est un rectangle défini par deux sommets opposés, le premier étant en haut à gauche et le second en bas à droite.
Les coordonnées en colonne sont des nombres compris entre 0 et 319 et en ligne entre 0 et 199.
Après l'exécution de cette instruction, les instructions de tracé graphique BOX, BOXF, LINE, PSET, CIRCLE, CIRCLEF et PAINT n'auront d'effet visible qu'à l'intérieur de la fenêtre.

Exemple:

```
WINDOW (100,100)-(200,180)
```

WRITE # n° de canal, liste de données
instruction

Ecrit dans le fichier, dont on précise le n° de canal. la liste des données.

Les chaînes de caractères sont entourées de guillemets et les différentes données sont séparées par des virgules. Après la dernière donnée, WRITE # écrit deux caractères: le saut de ligne et le Retour Chariot.

Dans le cas d'un fichier à accès direct, WRITE # écrit dans le tampon d'enregistrement. Le transfert de l'enregistrement dans le fichier doit alors se faire par PUT #.

Exemple:

```
WRITE #2, NOM$, TAUX
```

ZOOM
fonction

Donne la taille de la tortue active.

Le résultat est un nombre compris entre 0 et 255.
La taille normale d'une tortue au moment de sa définition par TURTLE est 16.

Exemple:

```
? ZOOM
32
```

ZOOM TO nombre
instruction

Fixe ou modifie la taille de la tortue active.

Si TO est présent, la taille est fixée de façon absolue à la valeur indiquée, sinon la taille est augmentée de cette valeur.
La taille d'une tortue peut varier de 0 à 255, mais la longueur d'un segment de la tortue ne doit pas dépasser 255.
Suivant la valeur du 2e paramètre de SHOW, la modification de la taille est immédiate (tortue libre) ou bien elle n'est effective qu'à

l'exécution de FWD ou TURTLE.

Exemple:

TURTLE 3: ZOOM TO 32 fixe la taille de la tortue nº 3 à 32
ZOOM 10 augmente de 10 la taille de la tortue active

Annexes

ANNEXE 1

Organisation d'une disquette

Cette annexe est destinée à ceux qui veulent en savoir un peu plus sur la manière dont le BASIC128 gère une disquette. Les informations que nous vous donnons ici sont un peu plus techniques, mais leur connaissance n'est pas du tout nécessaire, même pour faire un bon usage des fichiers.

Organisation physique

Une disquette est constituée de 40 pistes concentriques de 16 secteurs chacune. Chaque secteur contient 128 octets utiles sur une disquette en simple densité ou 255 octets utiles en double densité. Les 40 pistes sont numérotées de 0 à 39 et les 16 secteurs de 1 à 16.
Le découpage des pistes en 16 secteurs est fait au moment de l'initialisation de la disquette par le logiciel du BASIC. On dit alors que la sectorisation est logicielle; ce qui va de pair avec le fait que la disquette ou plus exactement le support en mylar ne possède qu'un seul trou d'index sur la circonférence. Ce trou unique sert à repérer la position du 1er secteur de chaque piste.

Contrairement à ce qu'on pourrait penser a priori, le secteur n°2 n'est pas celui qui suit immédiatement le secteur n°1 sur la piste. En fait, ils sont séparés par 7 secteurs, et ainsi de suite. Les secteurs sont donc entrelacés sur la piste avec exactement 7 secteurs entre deux numéros de secteurs successifs. Cette disposition a été adoptée pour permettre un minimum de traitement entre deux lectures ou deux écritures de secteurs successifs. En effet, il est très rare que l'ordinateur lise deux secteurs sans rien faire entre les deux lectures. Le facteur 7 a été choisi pour permettre la meilleure efficacité lors de la sauvegarde et du chargement d'un programme. Toutefois, si vous estimez qu'un autre facteur serait plus efficace pour les applications que vous envisagez, il est possible de le préciser au moment de l'initialisation:

```
DSKINI 0,2
```

initialisera la disquette située dans le lecteur n°0, en laissant 2 secteurs entre deux numéros de secteurs consécutifs.
Cette possibilité n'est vraiment intéressante que si le temps est pour vous un élément crucial. En effet, il vous faudra vous livrer à des mesures précises avant de pouvoir déterminer quel est le facteur optimal pour chacun de vos usages.

Le répertoire

Pour des raisons de commodité d'accès, le répertoire est situé sur la piste 20 de la disquette. On diminue ainsi la distance entre les données d'un fichier et les informations générales qui le concernent et qui sont inscrites dans le répertoire.
Le répertoire commence au secteur 3 et occupe une place de 14 secteurs.
Chaque fichier inscrit au répertoire a une zone réservée fixe de 32 octets. Cette zone contient les informations suivantes:

Nombre d'octets	*Contenu*
8	Nom du fichier, cadré à gauche, complété au besoin par les blancs à droite. Si le fichier a été supprimé par KILL, le premier octet contient 0. Si la zone du répertoire n'a pas encore été utilisée, le premier octet contient FF (hexadécimal).
3	Suffixe du nom du fichier (BAS,DAT,BIN,...), cadré à gauche, complété par des blancs à droite si nécessaire

1	Type du fichier: 0: programme BASIC 1: fichier de données BASIC 2: programme en langage machine 3: fichier de texte.
1	Type des données: 0: les octets contiennent du binaire FF: les octets contiennent des caractères codés en ASCII
1	le n° du premier bloc attribué au fichier (les blocs sont numérotés à partir de 0).
2	le nombre d'octets occupés dans le dernier secteur du fichier.
8	commentaire associé au fichier (lors de SAVE, COPY, NAME).
8	inutilisés.

Toutes les informations écrites dans le répertoire sont entièrement gérées par le BASIC. En particulier, le nombre d'octets occupés dans le dernier secteur ne peut être écrit qu'au moment de la fermeture: une bonne raison pour ne pas oublier CLOSE.

La table d'allocation mémoire de la disquette

Nous avons vu avec l'instruction DIR que la mémoire allouée à chaque fichier ne se comptait pas en secteurs mais en blocs de 8 secteurs soit 1 k-octets en simple densité et 2 k-octets en double densité.
Le secteur n° 2 de la piste 20 contient la table d'allocation mémoire, c'est-à-dire des informations sur les blocs de la disquette.
Dans cette table, chaque bloc est représenté par un octet. Attention: le 1er octet du secteur n'est pas utilisé. Le 1er bloc est représenté par le 2e octet, le 2e bloc par le 3e octet, etc.
La valeur de l'octet donne des indications sur l'occupation de bloc correspondant (1):

Valeur (hexadécimale)	Signification
entre 0 et 9F	Le bloc fait partie d'un fichier, l'octet contient le n° du prochain bloc du même fichier.
entre C1 et C8	Le bloc est le dernier d'un fichier, l'octet contient le nombre de secteurs du bloc occupés par le fichier auquel est ajouté la constante C0.
FE	Le bloc est réservé et ne peut pas être utilisé pour un fichier.
FF	Le bloc est libre.

Cette table d'allocation est mise à jour au fur et à mesure de l'écriture dans un fichier. Cependant, elle n'est pas mise à jour à chaque écriture dans un fichier. C'est la raison pour laquelle il est obligatoire d'utiliser UNLOAD lorsqu'on arrête une opération d'écriture de façon impromptue sans qu'il y ait eu de fermeture du fichier. UNLOAD recopie cette table sur la disquette et vous évite ainsi des ennuis si vous la remplacez par une autre dans le lecteur.

Comment voir le contenu de la disquette ? DSKI$ et DSKO$

L'envie vous est peut-être venue de voir un peu plus précisément ce que contient la disquette, secteur par secteur. Vous pouvez le faire avec une fonction qui est l'analogue de PEEK( ) dans la mémoire centrale :

```
DSKI$(0, 10, 3)
```

vous donne le contenu (une chaîne de 128 ou 255 caractères) du secteur 3 de la piste 10 du lecteur 0. Avec une telle fonction, vous pourrez examiner à loisir comment sont enregistrées les données dans un fichier à accès direct.
Cette fonction a son pendant DSKO$ qui vous permet d'écrire ce qui vous plaît dans le secteur que vous choisissez. Aussi, pour écrire la chaîne de caractères SCT$, de longueur 128 ou 255 octets maximum, dans le secteur 3 de la piste 10 du lecteur 0, vous faites :

```
DSKO$ 0, 10, 3, SCT$
```

En utilisant la fonction DSKI$( ) ou l'instruction DSKO$, n'oubliez pas que les pistes sont comptées de 0 à 39 et les secteurs de 1 à 16. Un dernier conseil: n'employez pas DSKO$ sur une disquette qui contient des programmes ou des données qui vous sont chers, une fausse manœuvre pourrait rapidement lui être fatale. Entraînez-vous d'abord sur des copies, c'est moins risqué!

Exécution automatique d'un programme

Si vous utilisez souvent le même programme, il est agréable de pouvoir lancer son exécution immédiatement après la mise en route. Pour ce faire, il vous faut sauvegarder le programme en question sous le nom "AUTO.BAT".
A la mise en route, quand vous choisirez l'option 2 du menu, l'interpréteur BASIC va chercher sur la disquette le programme AUTO.BAT, le charger et l'exécuter.
Pratique, non?

ANNEXE 2

Organisation de la mémoire utilisateur

Géographie générale de la mémoire

L'espace mémoire est organisé de la manière suivante:

adresse	
0000 3FFF	ROM BASIC extension ROM BASIC
4000 5FFF	mémoire écran forme mémoire écran couleur
6000 60FF	registres du moniteur
6100	

	6200
	tampons BASIC
	mémoire utilisateur non commutable
	pile système
9FFF	
A000	
	banques n°1 2, 3 4 5 6 avec extension 64 K
DFFF	
E000	
	moniteur et périphériques
FFFF	

La zone 6000 à 60FF est la page zéro du moniteur; elle contient tous les registres de travail.

La zone 6100 à 9FFF, non commutable, est utilisée par l'interpréteur BASIC pour les zones tampon, la manipulation des données et tous les pointeurs sur les variables et le programme.

La zone des banques de mémoire A000 à DFFF contient le programme, les variables et les données chaînes de caractères soit 32 k-octets disponibles.

Connaître l'espace mémoire disponible: FRE()

La fonction FRE permet de connaître les différents paramètres de l'espace mémoire restant:

FRE(2) donne le volume en octets de l'espace libre dans les banques de mémoire, c'est-à-dire libre pour programme et données.

FRE(1) donne le volume de l'espace libre dans la mémoire hors banques, c'est-à-dire libre pour les tampons et les pointeurs.

FRE(0) donne le volume de l'espace total disponible: FRE(0) = FRE(1) + FRE(2)

FRE(X$) donne le volume en octets de l'espace libre pour les chaînes de caractères.

Examen et modification d'octets en mémoire: PEEK et POKE

Il est possible de connaître la valeur d'un octet quelconque en mémoire par:

PEEK (adresse)

PEEK(&H8000) donne le contenu de l'octet situé à l'adresse 8000 (hexa).
La modification d'un octet se fait par:
POKE adresse,valeur
POKE &H8000,128 dépose la valeur décimale 128 (80 hexa) dans l'octet d'adresse 8000 (hexa).
Pour choisir la banque de mémoire à examiner, il faut utiliser l'instruction supplémentaire BANK.
BANK 2 oriente sur la banque n° 2 toutes les instructions qui agissent directement sur la mémoire.
Par la suite, l'exécution de PEEK(&HB000) donne le contenu de l'octet d'adresse B000 (hexa) de la banque n° 2.
A l'initialisation du micro-ordinateur, c'est la banque d'ordre le plus élevé qui est sélectionné, c'est-à-dire la banque n°2.
Pour choisir la banque d'ordre le plus élevé, sans avoir à se préoccuper du nombre de banques en présence, il faut faire:

BANK 0

Le résultat est le même qu'après l'initialisation.
La banque de mémoire sélectionnée par BANK est alors implicitement désignée pour les instructions et fonctions suivantes : PEEK, POKE, LOADM, SAVEM, EXEC, DEF USR, CLEAR.

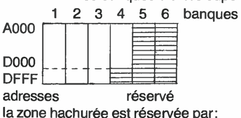
Réservation de zones mémoire : CLEAR

Toutes les modifications de l'espace disponible sont faites par l'instruction CLEAR.

1. *zone chaîne de caractères*
La modification la plus fréquente concerne l'espace disponible pour le stockage des chaînes de caractères.
Le volume de cet espace est fixé à l'initialisation à 300 caractères.
On le modifie avec le 1er argument de CLEAR.
CLEAR 2000 fixe à 2000 caractères cet espace

2. *réservation dans les banques de mémoire*
La réservation de mémoire est destinée à stocker des données en binaire ou des modules en langage machine. Cette réservation en

zone fixe permet de les utiliser depuis un programme en BASIC.
L'adresse de réservation moins un, est fixée par le 2e paramètre de CLEAR.
CLEAR ,&HCFFF réserve la mémoire depuis l'adresse &HD000 jusqu'à &HDFFF dans la banque de mémoire courante et réserve entièrement les banques d'ordre supérieur.



la zone hachurée est réservée par:

BANK 4: CLEAR,,&HCFFF avec l'extension mémoire 64 K

3. *réservation hors de la zone commutable*
Elle est fixée par le 4e argument de CLEAR.
Elle n'affecte que l'espace compris entre &H6100 et &H9FFF.
CLEAR ,,,&H8FFF réserve la zone comprise entre &H9000 et &H9FFF.

4. *réservation de caractères utilisateur*
Elle est fixée par le 3e paramètre de CLEAR.
Il suffit d'indiquer le nombre de caractères utilisateur maximum utilisés. Chaque caractère occupe huit octets.
CLEAR ,,10 fixe à 10 le nombre de caractères utilisateur et réserve la place nécessaire dans la zone des banques de mémoire.
Les réservations une fois fixées par CLEAR ne sont pas modifiées, même au chargement et à l'exécution d'un nouveau programme.
De nouvelles réservations ne sont possibles que par l'exécution d'une instruction CLEAR avec les arguments appropriés.

Sauvegarde et chargement d'une zone mémoire: SAVEM, LOADM

Ce couple d'instructions est destiné à conserver en fichier une zone précise de la mémoire, contenant des données ou un module en binaire.
La sauvegarde s'ecrit sous la forme:
SAVEM descr. de fichier, adr. début, adr. fin, adr. d'exécution

Le quatrième argument, adresse d'exécution, n'est utile que pour un programme en binaire, mais il est obligatoire.
Le chargement d'une zone s'écrit sous la forme:
LOADM descr. de fichier, déplacement ,R
Le deuxième argument, déplacement, quand il existe, indique le nombre d'octets dont il faut translater la zone mémoire par rapport à son adresse lors de la sauvegarde.
L'option ,R indique qu'il faut exécuter le module à l'adresse d'exécution sauvegardée dans SAVEM, en tenant compte du déplacement s'il y a lieu.

Exemple:

```
1Ø CLS                              'efface l'écran
2Ø BOXF (1ØØ, 1Ø) - (1Ø, 1ØØ), 1    'rectangle rouge
3Ø BOX (15Ø, 6Ø) - (11Ø, 2ØØ), 2    'rectangle vert
4Ø POKE &HE7C3, 1 OR PEEK (&HE7C3)
5Ø SAVEM"FORME", &H4ØØØ, &H5F3F, Ø
                                    'sauvegarde"FORME"
6Ø POKE &HE7C3, 254 AND PEEK (&HE7C3)
7Ø SAVEM"COULEUR", &H4ØØØ, &H5F3F, Ø
                                    'sauvegarde"COULEUR
```

ANNEXE 3

Représentation des variables en mémoire

La fonction VARPTR donne l'adresse de la variable donnée en argument et permet donc de connaître et de modifier son contenu.
Elle positionne automatiquement sur la banque de mémoire qui contient la variable en question.
Elle est utilisée le plus souvent pour interfacer un programme en BASIC avec un module en langage machine.

Variables entières

Ces variables sont représentées en binaire sur deux octets en complément à deux.
Le premier octet contient les bits de poids fort et le second les bits de poids faible.
Le bit de poids le plus fort du premier octet représente le signe:
- 0 nombre positif ou nul
- 1 nombre négatif

Les quinze autres bits permettent donc des valeurs absolues jusqu'à 2^15 soit 32 768.

Les nombres entiers s'étendent donc de −32768 à 32767.
La fonction VARPTR rend l'adresse du premier octet.

Exemple:

```
X%=12
? PEEK(VARPTR(X%)),PEEK(VARPTR(X%)+1)
0     12
```

Variables réelles ou simple précision

Ces variables sont représentées sous forme flottante (mantisse + exposant) sur 4 octets.

Le premier octet contient l'exposant.
L'exposant est noté en binaire signé et décalé de &H80.
Ainsi &H81 correspond à 2^1
et &H7E correspond à 2^−2.
Un exposant nul signifie un nombre nul.

Le 2e octet contient les bits de poids fort de la mantisse.
Le 3e octet contient le bit de poids moyen de la mantisse.
Le 4e octet contient le bit de poids faible de la mantisse.
Le bit de poids le plus fort de la mantisse représente le signe du nombre. Les autres bits représentent la mantisse en binaire, sauf le 1er bit de celle-ci toujours égal à 1, et qui donc est omis. La valeur de la mantisse peut donc s'écrire, en base 2:
0.1mantisse
En conséquence, le plus petit nombre positif vaut 2.9874 E-39 et le plus grand vaut 1.7014 E+38.
La fonction VARPTR rend l'adresse du premier octet, donc de l'exposant.

Exemples:

```
NB=3
ADNB=VARPTR(NB)
FOR I=0 TO 3: PRINT PEEK(ADNB+I);: NEXT
130 64 0 0
```

soit un exposant égal à 2^(130−128)=2^2

et une mantisse égale à 0.11000000 00000000 00000000

```
NB=-1
ADNB=VARPTR(NB)
FOR I=0 TO 3: PRINT PEEK(ADNB+I);: NEXT
129 128 0 0
```

soit un exposant égal à 2^(129−128)=2^1
et une mantisse égale à 0.10000000 00000000 00000000
et un signe négatif

Variables en double précision

Ces variables sont représentées sous forme flottante, de la même manière que les variables en simple précision mais sur 8 octets.
Le premier octet contient l'exposant, codé comme en simple précision, les sept suivants contiennent la mantisse.
Les valeurs minimum et maximum sont les mêmes que celles des variables en simple précision.

Variables chaînes de caractères

Ces variables sont constituées de deux parties, un descripteur sur trois octets et le contenu de la chaîne proprement dit.
Le premier octet du descripteur contient la longueur de la chaîne.
Les deux octets suivants contiennent l'adresse logique de la chaîne dans la zone chaîne de caractères.
L'adresse de la zone chaîne est contenue aux adresses &H6122 et &H6123. Le numéro de la banque mémoire qui contient la zone chaîne est indiqué en &H6124.
L'adresse physique de la chaîne est obtenue en ajoutant à l'adresse de la zone chaîne l'adresse logique de la chaîne.
Si l'adresse physique dépasse &HE000, il faut y retrancher &HE000 et y ajouter &HA000 pour obtenir la bonne valeur, ceci pour tenir compte d'un chevauchement de la zone chaîne entre deux banques de mémoire.
La fonction VARPTR rend l'adresse du premier octet du descripteur.

Exemple:

```
5 ' CAS SIMPLE SANS CHEVAUCHEMENT
10 CH$="ABCDEF"
20 ADCH=VARPTR(CH$)
30 LCH=PEEK(ADCH): ADLOG=256*PEEK(ADCH+1)+PEEK(ADCH+2)
40 PRINT LCH,ADLOG
50 ADZCH=256*PEEK(&H6122)+PEEK(&H6123)
60 BCH=PEEK(&H6124)
70 PRINT HEX$(ADZCH),BCH
80 BANK BCH
90 PRINT CHR$(PEEK(ADZCH+ADLOG))
RUN
6       0
DED3    6
```

ANNEXE 4

Modules et fonctions en langage machine

Cette annexe intéresse surtout ceux qui ont quelques notions de programmation du microprocesseur 6809 en langage machine. Elle suppose une connaissance minimale de ce microprocesseur.
Il existe deux solutions pour utiliser le langage machine : les modules exécutables par EXEC, et les fonctions utilisateur USR.

L'instruction EXEC avec paramètres

La première chose à faire avant tout est de réserver la place nécessaire aux instructions machine. On utilise pour cela CLEAR qui réserve la zone mémoire utile:

```
CLEAR, &HDEFF
```

fixe l'adresse la plus haute du BASIC à &HDEFF, dans la banque courante, par défaut dans la dernière. Il reste donc 256 octets libres jusqu'au fond de la banque de mémoire en &HDFFF.

La deuxième chose est d'y déposer les instructions par des POKE, ou bien par LOADM si ces instructions ont été conservées en fichier, prêtes à être exécutées.

Enfin, on exécute le module en langage machine par:

```
EXEC &HDF00
```

si l'adresse de début du module est en &HDF00.

Toutefois, avant d'écrire le module en langage machine, il faut respecter certaines règles si l'on veut revenir à la suite dans le programme Basic:
— le module doit se terminer par RTS,
— les registres S (*Stack*) et DP (*Direct Page*) ne doivent pas avoir été modifiés. Pour pouvoir les utiliser dans le module, il faudra les sauvegarder à l'entrée et les récupérer à la sortie.

Il est possible de passer des paramètres au module en langage machine. Ces paramètres doivent suivre l'adresse, séparés par des virgules.
La récupération des paramètres dans le module se fait en appelant l'intrepréteur Basic.
Les points d'entrées utiles pour analyser les paramètres sont les suivants:

3FEB LITINT	lit un entier signé, résultat dans le registre X
3FEE LITADR	lit un entier non signe (adresse), résultat dans le registre X
3FF1 LITSGN	lit un réel simple précision. le registre X pointe sur le résultat
3FF4 LITSTR	lit une chaîne, en sortie le registre B contient la longueur, le registre X pointe sur le résultat
3FF7 FRMEVL	lit une donnée quelconque, en sortie on obtient le même résultat qu'avec une fonction USR, voir plus loin
3FFA PTRGET	rend l'adresse du paramètre, en sortie le registre A contient le numéro de banque, le registre X pointe sur le contenu, l'octet VALTYP ($6105) contient le type du résultat (2, 3, 4 ou 8)

3FFD IFEND	teste la fin de la ligne, en sortie le drapeau Z du registre d'état est à 1 si la fin est atteinte

Les fonctions USRn

On peut définir dix fonctions utilisateur en langage machine par l'instruction:

DEF USRn = adresse
(n est un chiffre (de 0 à 9)
adresse désigne l'adresse mémoire à laquelle commence la fonction.)

L'appel de la fonction se fait alors comme pour une fonction normale:

variable = USRn (expression)

L'avantage principal des fonctions USR est qu'elles permettent de passer des valeurs et d'en retourner au programme Basic par le résultat fourni.
Nous allons examiner comment ces valeurs sont passées aux fonctions utilisateur.
A l'appel de la fonction, l'accumulateur A contient le type de la donnée:
2 : nombre entier
3 : chaîne de caractères
4 : nombre en simple précision
8 : nombre en double précision

le registre d'index X pointe sur la valeur:
2,X : nombre entier
0,X : nombre en simple ou double précision
0,X : descripteur pour une chaîne

Dans le cas où la donnée est une chaîne de caractères, l'accumulateur B contient la longueur de la chaîne, le registre U pointe sur le 1er caractère de la chaîne.
Il faut remarquer que, sauf pour les chaînes de caractères,

l'accumulateur A contient en même temps la longueur de la donnée en octets.
Rappelons que :
— les entiers sont codés sur deux octets : 1er octet poids forts, 2e octet poids faibles,
— les nombres en simple précision sont codés sur quatre octets : le premier contient l'exposant, les suivants la mantisse, poids forts en tête,
— les nombres en double précision sont codés sur huit octets : le premier contient l'exposant, les suivants la mantisse.
— le descripteur d'une chaîne de caractères comporte trois octets : le premier indique la longueur de chaîne, les deux autres adresse du 1er caractère.
On peut donc atteindre la longueur de la chaîne soit par l'accumulateur B, soit par le descripteur.
La valeur (nombre ou descripteur de chaîne) est toujours déposé au même endroit (FAC).

A la sortie du module assembleur, il faut également rendre les contenus de l'accumulateur A et du registre X compatibles avec le type de la variable qui va recevoir la valeur de sortie.
En sortie, X doit donc pointer sur la même adresse qu'à l'entrée.
Dans tous les cas, les registres S et DP doivent être rendus avec les mêmes contenus qu'à l'entrée.

Exemple: Ajouter 256 à un entier.
Après avoir réservé la place (largement suffisante) en ligne 20, on lit les différentes instructions machine, introduites en DATA, et on les dépose en mémoire (lignes 40 à 70).
Pour pouvoir mettre un nombre variable d'instructions en DATA, la dernière valeur, fictive, est supérieure à 255 (&H100) et sert au test de fin de lecture.
Le module lui-même ne comporte que deux instructions machine (on ne peut faire plus simple) :

```
6C 02 INC 2,X; incrémenter en (X)+2
39 RTS ; retour
```

L'incrémentation de l'octet de poids fort revient à ajouter 256. Notez que pour les entiers, la valeur n'est pas en (X) mais (X)+2.
La nouvelle variable R% reçoit alors la nouvelle valeur qui est affichée avec la précédente.

```
10 ' AJOUTER 256
20 CLEAR ,&HDEFF
30 DEFUSR1=&HDF00
40 ADR=&HDF00
50   READ OCT : IF OCT>255 THEN 80
60   POKE ADR,OCT
70   ADR=ADR+1 : GOTO 50
80 ' UTILISATION
90 N%=10
100 R%=USR1(N%)
110 PRINT N%,R%
120 DATA &H6C,&H02,&H39,&H100
```

ANNEXE 5

Liste des mots réservés

ABS
AND
ASC
ATN
ATTRB
AUTO

BACKUP
BANK
BEEP
BOX

CDBL
CHAIN
CHR$

CINT
CIRCLE
CLEAR
CLOSE
CLS
COLOR
COMMON
CONSOLE
CONT
COPY
COS
CRUNCH$
CSNG
CSRLIN
CVD

CVI
CVS

DATA
DEF
DEFDBL
DEFINT
DEFSNG
DEFSTR
DELETE
DENSITY
DEVICE
DIM
DIR
DO

DSKI$
DSKF
DSKO$

ELSE
END
EOF
EQV
ERL
ERR
ERROR
EVAL
EXEC
EXIT
EXP
FIELD
FILES
FIX
FKEY$
FN
FOR
FRE

GET
GO
GR$

HEAD
HEX$

IF
IMP
INKEY$
INMOUSE
INPEN
INPUT
INSTR
INT
INTERVAL

KEY
KILL

LEFT$
LEN
LET
LINE
LIST
LOAD
LOC
LOCATE
LOF
LOG
LOOP
LSET

MAX
MERGE
MKD$
MKI$
MKS$
MID$
MIN
MOD
MOTOR
MOUSE
MTRIG

NAME
NEW
NEXT
NOT

OCT$
OFF
ON
OPEN
OR

PAINT
PALETTE
PATTERN
PEEK
PEN
PLAY

POINT
POKE
POS
PRINT
PSET
PTRIG
PUT

READ
REM
RENUM
RESET
RESTORE
RESUME
RETURN
RIGHT$
RND
ROT
RSET
RUN

SAVE
SCREEN
SEARCH
SGN
SHOW
SKIPF
SPACE$
SPC(
SQR
STEP
STICK
STOP
STRIG
STRING$
STR$
SUB
SWAP

TAB(
TAN
THEN

TO
TRACE
TROFF
TRON
TURTLE

UNLOAD
UNMASK
USING
USR

VAL
VARPTR

WAIT
WINDOW
WRITE

XOR

ZOOM

ANNEXE 6

Messages et codes d'erreurs

1 **Next Without For**

Une instruction NEXT a été rencontrée avant le FOR correspondant.

2 **Syntax Error**

Erreur de syntaxe : l'instruction n'est pas connue ou mal rédigée.

3 **Return Without Gosub**

L'instruction RETURN a été rencontrée sans qu'il y ait appel par GOSUB.

4 **Out Of Data**

Il n'y a plus assez de données dans les lignes de DATA pour le READ à exécuter.

5 **Illegal Function Call**

Une fonction ou une instruction est appelée avec des valeurs interdites.

6 **Overflow**
La valeur numérique obtenue est trop grande.

7 **Out Of Memory**
Il n'y a plus assez de place en mémoire centrale.

8 **Undefined Line Number**
On ne peut pas faire de GOTO ou de GOSUB à une ligne qui n'existe pas.

9 **Subscript Out Of Range**
L'indice d'un élément de tableau dépasse la valeur maximum ou est négatif.

10 **Duplicate Definition**
Il n'est pas possible de déclarer deux fois le même tableau.

11 **Division By Zero**

12 **Illegal Direct**
L'instruction ne peut pas être utilisée en mode direct.

13 **Type Mismatch**
Il n'est pas possible de mettre un nombre dans une variable chaîne et inversement.

14 **Out Of String Space**
Il n'y a plus assez de place en mémoire centrale dans la zone des chaînes de caractères.

15 **String Too Long**
Une chaîne de caractères ne peut pas dépasser 255 caractères.

16 **String Formula Too Complex**
Expression portant sur les chaînes de caractères trop compliquée à traiter.

17 **Can't Continue**
La commande CONT ne permet pas de poursuivre l'exécution du programme.

18 **Undefined User Function**
La fonction USR utilisée n'a pas été définie.

19 **No Resume**
Il n'y a pas d'instruction RESUME dans la partie qui traite les erreurs (ON ERROR GOTO).

20 **Resume Without Error**
L'instruction RESUME est rencontrée alors qu'il n'y a pas eu d'erreur.

21 **Undefined Error**
Erreur non définie (simulée par ERROR).

22 **Missing Operand**
Il manque un opérande dans une opération comme addition, soustraction,...

23 **For Without Next**
Aucun NEXT n'a été rencontré après l'exécution de FOR.

24 **Can't Exit**
L'instruction EXIT est utilisée en dehors d'une boucle.

25 **Do Without Loop**
Aucun LOOP n'a été rencontré après l'exécution de DO.

26 **Loop Without Do**
L'instruction LOOP est rencontrée alors que le DO correspondant n'a pas été exécuté.

27 **Ram Error**
Erreur d'accès à la mémoire.

Erreurs portant sur les fichiers

50 **Bad File Number**
Ce numéro de fichier n'est pas utilisé.

51 **Bad File Mode**
Le mode d'utilisation du fichier est incorrect.

52 **File Already Open**
Un fichier déjà ouvert ne peut pas être ouvert à nouveau.

53 **Device I/O Error**
Problème matériel d'accès au périphérique.

54 **Input Past End**
Il n'est pas possible de lire au-delà de la fin du fichier.

55 **Bad File Descriptor**
Le descripteur de fichier n'est pas correct.

56 **Direct Statement In File**
Le fichier en cours de chargement contient une commande d'exécution directe.

57 **File Not Open**
Le fichier sur lequel on veut lire ou écrire n'est pas ouvert.

58 **Bad Data In File**
Les données lues dans un fichier ne sont du type de celles qui sont attendues.

59 **Device In Use**
Le périphérique est déjà en fonctionnement.

60 **Device Unavailable**
Le périphérique n'est pas disponible.

61 **Protected Program**
Le programme est protégé et ne peut pas être listé ou sauvegardé.

62 **File Not Found**
Le fichier n'existe pas sur la disquette.

63 **Disk Full**
Il n'y aplus de place disponible sur la disquette.

64 **Too Many Open Disk Files**
On ne peut pas ouvrir plus de fichiers qu'il n'est indiqué dans l'instruction FILES.

65 **Directory Full**
Il n'y a plus de place sur le répertoire.

66 **File Already Exists**
Le nom du nouveau fichier dans NAME correspond à un fichier existant.

67 **Field Overflow**
La somme des longueurs des variables de champ dans un FIELD dépasse la longueur de l'enregistrement.

68 **String Fielded**
Une variable de champ ne peut être affctée que par LSET ou RSET.

69 **Bad Record Number**
Le numéro d'enregistrement dans PUT ou GET n'est pas possible.

70 **Bad File Structure**
Les indications concernant les blocs du fichier dans la table d'allocation ne sont pas cohérentes.

71 **No Disk**
Le lecteur de disquettes est vide.

72 **Disk Write Protected**
La disquette est protégée en écriture.

73 **Out Of Fielded Buffers**
Il n'y a plus assez de place dans la zone réservée aux tampons des fichiers à accès direct.

74 **End Of Record**
La lecture ou l'écriture dans un fichier à accès direct dépasse la fin de l'enregistrement.

75 **Verification Failure**
La relecture après écriture (VERIFY ON) indique une différence.

76 **Unreadable diskette**
La disquette n'est pas formatée.

77 **Density Mismatch**
La densité ne correspond pas.

78 **Bad Picture**
L'image ne peut pas être chargée ou sauvegardée.

ANNEXE 7

CODE ASCII

Code décimal	
000	NUL
001	
002	STOP touche STOP clavier
003	BREAK (CNT C)
004	
005	
006	
007	SONNETTE
008	BS (Recul arrière) ← clavier
009	HT (Tabulation H) → clavier
010	LF (Saut de ligne) ↓ clavier
011	VT (Tabulation V) ↑ clavier
012	FF (Saut de page) RAZ clavier
013	CR (Retour à la ligne) ENTREE
014	SO Mode semi-graphique
015	SI Mode alphanumérique
016	
017	DC1 Clignotement curseur
018	DC2 Répétition

Code décimal	
019	
020	DC4 Arrêt curseur
021	
022	SS2 Touche caractères accentués
023	
024	CAN Efface la fin de la ligne
025	
026	
027	ESC Appel d'une séquence
028	INS (INS clavier)
029	DEL (EFF clavier)
030	RS Touche clavier
031	US Séparateur d'article
032	Espace
033	!
034	''
035	#
036	$
037	%
038	&
039	'
040	(
041	)
042	*
043	+
044	,
045	-
046	.
047	/
048	0
049	1
050	2
051	3
052	4
053	5
054	6
055	7
056	8
057	9
058	:
059	;

Code décimal	
060	<
061	=
062	>
063	?
064	@
065	A
066	B
067	C
068	D
069	E
070	F
071	G
072	H
073	I
074	J
075	K
076	L
077	M
078	N
079	O
080	P
081	Q
082	R
083	S
084	T
085	U
086	V
087	W
088	X
089	Y
090	Z
091	[
092	\
093	]
094	↑
095	_
096	`
097	a
098	b
099	c
100	d
101	e
102	f
103	g
104	h
105	i
106	j
107	k
108	l
109	m
110	n
111	o
112	p
113	q
114	r
115	s
116	t
117	u
118	v
119	w
120	x
121	y
122	z
123	{
124	|
125	}
126	~
127	■

ANNEXE 8

Index thématique

Cet index thématique vous permet de retrouver une instruction Basic dans le domaine qui vous intéresse. Le numéro de page correspondant à l'instruction figure dans l'index général.

Commandes et instructions générales

Fonctions numériques

ABS
ATN
CDBL
CINT
COS
CSNG
EXP
FIX
INT
LOG
MAX
MIN
RND
SGN
SIN
SQR
TAN

Fonctions sur chaînes de caractères

ASC
CHR$
INSTR
LEFT$
LEN
MID$
RIGHT$
SPACE$
STR$
STRING$
VAL

Fonctions diverses

BANK
CRUNCH$
EVAL
FRE
HEX$
INKEY$
INPUT$
OCT$
PEEK
USR
VARPTR

FICHIERS

Instructions

BACKUP
CLOSE
COPY
DENSITY
DEVICE
DIR
DIRP
DSKINI
DSKO$
FIELD
FILES
GET#
INPUT#
KILL
LINEINPUT#
LSET
MOTORON
MOTOROFF
NAME
OPEN
PRINT#
PRINT# USING
PUT#
RSET
SKIPF
UNLOAD
VERIFYON
VERIFYOFF
WRITE#

Fonctions

CVD
CVI

CVS
DSKF
DSKI$
EOF
LOC
LOF
MKD$
MKI$
MKS$
POS

AFFICHAGE CARACTERE

Instructions

ATTRB
CLS
COLOR
CONSOLE
DEFGR$
LOCATE
PRINT
PRINT USING
SCREEN
UNMASK

Fonctions

CSRLIN
GR$
POS
SPC
TAB
SCREEN

AFFICHAGE GRAPHIQUE

Instructions

BOX
BOXF
CIRCLE
CIRCLEF
GET
LINE
PAINT
PATTERN
PSET
PUT
WINDOW

Fonction

POINT

TORTUES

Instructions

FWD
HEAD
INPUTTURTLE
ROT
SHOW
TRACE
TURTLE
ZOOM

Fonctions

HEAD
ROT
SHOW
TRACE
ZOOM

AUTRES ENTREES-SORTIES

Instructions

INPEN
INPUTPEN
ONPEN ...GOTO
ONPEN ...GOSUB
PEN
PLAY

Fonctions

PTRIG
STICK
STRIG

ANNEXE 9

Index général

Le premier renvoi de page concerne la partie Réferences. le deuxième, la partie Initiation.

TABLE DES MATIÈRES



Achevé d'imprimer en France par Pollina, 85400 Luçon - N° 7507
Dépôt légal : Octobre 1985